To Each Language Its Tokenizer: Modular Tokenizers for Efficient Multilingual LLMs

Franck Signe[1,2], Hippolyte Pilchen[1,3], François Yvon[2], Édouard Grave[1]

[1]Kyutai, Paris
[2]Sorbonne Université, CNRS, Institute of Intelligent Systems and Robotics, Paris
[3]Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, Grenoble

Correspondence: franck.signe-talla@kyutai.org

Abstract

Multilingual Large Language Models (LLMs) traditionally rely on a single vocabulary shared by all supported languages, which can lead to uneven compression across them. Moreover, their large embedding and output matrices increase memory usage and slow inference, notably for small-scale models. It is also wasteful as models are often used for only a subset of languages. To address these issues, we introduce a modular framework for multilingual model training. First, we propose methods to learn large modular BPE and Unigram tokenizers that enable extraction of subtokenizers tailored to any language subset. These subtokenizers achieve compression on par with monolingual tokenizers and improve cross-lingual fairness. Second, we design a pretraining strategy that samples subtokenizers to form batches, restricting predictions to the relevant vocabulary subset and allowing efficient training despite a large vocabulary. This supports efficient inference with any combination of language-specific vocabularies. Therefore, it reduces memory usage and speeds up inference in models without sacrificing performance.[1]

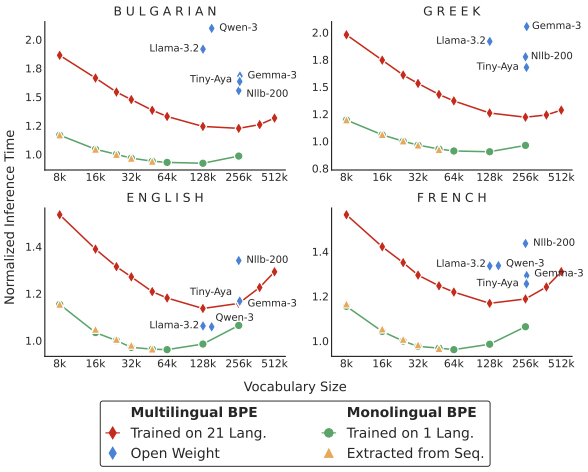


Figure 1: Normalized inference time on FLORES 200 for a 2.5B non-embedding parameter model using monolingual and multilingual tokenizers of varying sizes. $24k$-token monolingual tokenizers are used as reference. For each size, *Extracted* denotes monolingual tokenizers derived from a sequentially trained multilingual BPE model (§ 4.2), with a per-language budget of that size across 21 languages. Baseline open-weight tokenizers cover all reported languages.

1 Introduction

Tokenization is the interface between raw text and the embedding space of Large Language Models (LLMs), directly influencing vocabulary coverage, sequence length, and computational efficiency. In multilingual settings, tokenizers must cover diverse scripts and linguistic patterns, often requiring large vocabularies. In smaller models, the embedding matrix can account for a disproportionate share of total parameters — for example, Gemma 2 2.6B (Team et al., 2024) devotes 22% of its parameters to a 256k-token vocabulary, even with shared input and output embeddings. This is particularly wasteful in resource-constrained settings, where models are often deployed for specific language(s) and the full multilingual output matrix need not be loaded, leaving most tokens redundant and wasting memory and compute. Beyond computational inefficiencies, standard multilingual tokenizers also induce compression disparities across languages: identical content can yield drastically different tokenization lengths depending on the language (Petrov et al., 2023), leading to unequal effective context windows and higher computational costs for non-English text. Together, these issues make inference with large multilingual tokenizers covering many languages slower and more expensive than with language-specific ones, as illustrated in Fig. 1.

To address both challenges at pretraining, we propose two complementary contributions. First, a method to construct a modular multilingual tok-

[1]The code is available at https://github.com/kyutai-labs/modular-tokenization.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 21534–21563
October 24-29, 2026 ©2026 Association for Computational Linguistics

enizer from which specialized subtokenizers can be extracted, each covering a specific language or subset of languages, with compression rates comparable to independently trained monolingual tokenizers, provided for both BPE (Gage, 1994; Sennrich et al., 2016) and Unigram (Kudo, 2018). Second, a pretraining procedure that trains the model to operate with any combination of subtokenizers tailored to the language(s) of interest, allowing irrelevant tokens to be excluded from the output matrix at inference. By maintaining high compression for target languages, this yields faster inference than standard multilingual tokenizers while retaining competitive performance on multiple-choice evaluation, summarization, and translation.

2 Related Work

Unfairness in Tokenization. Multilingual tokenizers often compress languages unevenly: identical content translated from English can yield widely varying token lengths, with some low-resource languages requiring up to 15 times more tokens (Petrov et al., 2023), raising computational costs and penalizing them in token-based pricing schemes (Ahia et al., 2023). These disparities are not limited to multilingual settings: Arnett et al. (2025) show that monolingual BPE tokenizers trained under identical conditions (parallel content-matched corpora, identical vocabulary sizes and training procedures) still exhibit different compression rates across languages. They attribute this largely to whitespace-based pre-tokenization which prevents merges across word boundaries, a limitation that superword tokenizers (Liu et al., 2025; Schmidt et al., 2025) address by allowing merges across whitespace boundaries.

To address these disparities, Foroutan et al. (2026) propose a BPE variant that selects, at each step, the merge maximizing compression for the worst-compressed language. Other work leverages byte-level models: Ahia et al. (2024) propose adaptive gradient-based tokenization to group bytes, and Limisiewicz et al. (2024) modify UTF-8 representations based on morphology to balance compression across languages. Zheng et al. (2021) explicitly allocate vocabulary capacity across languages, increasing the share of underrepresented languages. Chung et al. (2020); Liang et al. (2023) group linguistically similar languages to reduce harmful vocabulary sharing, training tokenizers for each cluster before merging them for multilingual coverage, and use these different tokenizers to train encoder-based models. In contrast, our approach enables flexible inference with any combination of specialized vocabularies, without sacrificing performance, notably on generation tasks.

Vocabulary Adaptation. The computational efficiency of a pretrained model on a target domain depends heavily on vocabulary suitability, motivating two adaptation strategies: vocabulary expansion to reduce sequence length, and vocabulary reduction to lower computational overhead.

Expansion methods append language-specific tokens to pretrained vocabularies (Tejaswi et al., 2024; Csaki et al., 2024; Cui et al., 2024; Kiulian et al., 2025; Fujii et al., 2024; Lin et al., 2024; Dagan et al., 2024), achieving more compact representations, lower tokenizer fertility, and higher throughput. Usually, a dedicated tokenizer is trained on the target distribution and its new tokens are appended to the original vocabulary, but existing merge rules can prevent these tokens from being formed, making this approach suboptimal. Purason et al. (2026) address this by continuing the training of the original tokenizer on the target distribution, refining its merge table rather than simply appending new tokens.

Reduction methods adapt vocabularies to target language(s) by pruning irrelevant tokens (Ushio et al., 2023; Williams and Aletras, 2025; Pilana Liyanage and Yvon, 2026; Zhang et al., 2026) or replacing them with domain-specific ones (Nakash et al., 2025; Chijiwa et al., 2026), reducing memory and improving inference speed. The latter requires careful embedding initialization (Minixhofer et al., 2022; Gee et al., 2022; Dobler and de Melo, 2023; Samuel et al., 2025; Nakash et al., 2025), and both require a fine-tuning step to maintain performance. Unlike prior work, we tackle these issues during pretraining rather than post-hoc, enabling vocabulary reduction gains to be realized effortlessly at inference.

3 Inefficiencies of Multilingual Tokenizers

A tokenizer $\mathcal{T}$ is formally defined as a tuple $(\mathcal{V}, \tau, \delta)$, where $\mathcal{V}$ is a finite vocabulary of tokens (Zouhar et al., 2023). The tokenization function $\tau : \Sigma^* \to \mathcal{V}^*$ maps a string S defined over a finite alphabet Σ to a sequence of tokens, which we denote for simplicity as $\mathcal{T}(S) = \tau(S)$. The detokenization function $\delta : \mathcal{V}^* \to \Sigma^*$ maps a token

sequence back to a string over the original alphabet, with most modern tokenizers satisfying the *lossless constraint* $\delta(\mathcal{T}(S)) = S$ for all $S \in \Sigma^*$.

Tokenizer Training. BPE tokenizers are trained with the Hugging Face tokenizers library[2] and Unigram tokenizers with SentencePiece,[3] both using whitespace pre-tokenization with byte-level fallback and no normalization. Monolingual tokenizers are trained on language-specific CommonCrawl subsets;[4] multilingual ones on the concatenation of these subsets, ensuring identical per-language data exposure in both settings.

3.1 Monolingual Tokenizers Better Compress than Multilingual Ones

Multilingual LLMs require tokenizers with broad language coverage to handle diverse scripts and linguistic structures, but larger vocabularies increase the computational cost of the output layer, forcing tokenizers to operate under limited capacity. This constraint disproportionately affects underrepresented languages, which tend to be poorly compressed (Zheng et al., 2021). To quantify the compression efficiency of a tokenizer on a given language $\mathcal{L}$, we use the **Normalized Sequence Length (NSL)** (Dagan et al., 2024), defined as the ratio of tokenized sequence length to that of a reference tokenizer. For a dataset $\mathcal{D}$ with M examples in language $\mathcal{L}$, NSL is defined as:

$$\text{NSL} = \frac{\sum_{i=1}^{M} \text{length}\big(\mathcal{T}(\mathcal{D}[i])\big)}{\sum_{i=1}^{M} \text{length}\big(\mathcal{T}^{l}_{\text{ref}}(\mathcal{D}[i])\big)}, \qquad (1)$$

where $\mathcal{T}^{l}_{\text{ref}}$ is a monolingual reference tokenizer of fixed vocabulary size trained exclusively on $\mathcal{L}$.

Tab. 1 reports NSL on FLORES-200 (devtest, 1012 lines) for multilingual tokenizers relative to monolingual reference with 24k tokens. We focus on BPE, since the open-weight tokenizers are BPE-based, ensuring a uniform evaluation; the same conclusions hold for Unigram tokenizers (see Appendix A.3.3). Open-weight tokenizers — TinyAya (Salamanca et al., 2026), Nllb-200 (Costa-jussà et al., 2024), Gemma-3 (Team et al., 2025), Llama-3 (Grattafiori et al., 2024), and Qwen-3 (Yang et al., 2025) — achieve compression comparable to monolingual references for English but underperform for most other languages.

[2] https://huggingface.co/docs/transformers/main_classes/tokenizer

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

[4] https://data.commoncrawl.org/crawl-data/CC-MAIN-2024-10/index.html

Model	Size	NSL (↓) BG	EL	EN	FI	FR
TinyAya	261k	1.38	1.42	0.98	1.35	1.06
NLLB	256k	1.33	1.51	1.14	1.30	1.21
Gemma3	262k	1.42	1.73	0.98	1.54	1.09
Llama3	128k	1.78	1.78	0.98	1.83	1.24
Qwen3	152k	1.92	3.90	0.96	1.82	1.22
Multi 21	128k	1.16	1.17	1.06	1.14	1.09
	256k	1.04	1.04	0.98	1.02	1.01

Table 1: NSL relative to a 24k-token monolingual BPE reference, for open-weight tokenizers and a standard multilingual BPE jointly trained on our 21 languages (*Multi 21*). More languages in Appendix A.3.2.

By spreading a fixed vocabulary across many languages, these tokenizers end up providing poor compression for the majority of languages they aim to support.

Our evaluation set is deliberately conservative: 21 European languages (Appendix A.2), medium- to high-resource and well covered by open-weight multilingual tokenizers — unfairness thus appears even in a setting that favors these tokenizers, and it widens sharply on low-resource languages (Tab. 2). Covering fewer languages leaves more vocabulary per language: *Multi 21*, a standard BPE trained on our 21 languages only, comes within 4% of monolingual compression at 256k tokens — a size comparable to the largest open-weight tokenizers in Tab. 1. This *trade-off* between coverage and per-language compression motivates specialized tokenizers over a single large multilingual one.

While better compression allows more data to be processed for the same compute budget at training, its impact on model quality remains contentious: some work finds a positive correlation with downstream performance (Gallé, 2019; Goldman et al., 2024), while others find that increased compression fails to yield performance gains (Schmidt et al., 2024; Dagan et al., 2024). In this work, we therefore pursue improved compression primarily for efficiency rather than performance gains.

3.2 Monolingual Tokenizers Enable Cheaper Inference than Multilingual Ones

The previously discussed *trade-off* in multilingual coverage often forces large-coverage tokenizers to adopt extensive vocabularies to maintain competitive inference speeds. As shown in Fig. 1, initial vocabulary expansions bring substantial speed gains, but these quickly plateau as marginal compression

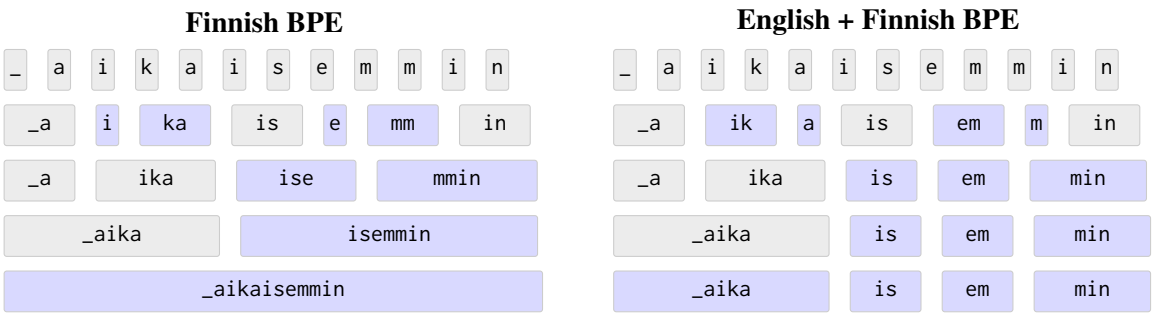


Figure 2: **Comparison of BPE merge hierarchies.** The Finnish-only tokenizer (left) forms the token *_aikaisemmin*. In contrast, the joint English–Finnish tokenizer (right), where Finnish rules are appended to English ones, produces suboptimal units: early English merges *e+m* and *m+in* prevent *_aikaisemmin* from forming despite its presence in the vocabulary. Our *Sequential training* mitigates this by continuing training the English tokenizer on Finnish data, creating merges *em+min*, *is+emmin*, and *_aika+isemmin* that ultimately recover the token *_aikaisemmin*.

improvements are offset by a growing output-layer cost. This cost is especially pronounced in small-scale models, where the embedding and output matrices represent a large share of total parameters, amplifying both memory overhead and inference latency. Fig. 1 suggests that monolingual tokenizers sidestep this issue, achieving faster inference with much smaller vocabularies.

4 Building Modular Tokenizers

Given a global tokenizer $\mathcal{T} = (\mathcal{V}, \tau, \delta)$, we define a **subtokenizer** $\mathcal{T}_s = (\mathcal{V}_s, \tau_s, \delta_s)$ where $\mathcal{V}_s \subseteq \mathcal{V}$. To leverage the advantages of monolingual tokenizers within a multilingual framework, we aim to design $\mathcal{T}$ covering languages $\{\mathcal{L}_1, \ldots, \mathcal{L}_N\}$ that satisfies: **(i) Extractability:** for each $\mathcal{L}_i$, a specialized subtokenizer $\mathcal{T}_i = (\mathcal{V}_i, \tau_i, \delta_i)$ can be explicitly derived; **(ii) Compression Efficiency:** each $\mathcal{T}_i$ matches the compression of a tokenizer trained only on $\mathcal{L}_i$; and **(iii) Compositionality:** for any subset $\{\mathcal{L}_1, \ldots, \mathcal{L}_k\}$, a unified subtokenizer $\mathcal{T}_s$ can be built via $\mathcal{V}_s = \bigcup_{i=1}^{k} \mathcal{V}_i$, maintaining competitive compression across all included languages.

4.1 UnigramLM Tokenizer

The UnigramLM tokenization function assigns to each token in $\mathcal{V}$ a precomputed probability, assuming independence between token occurrences (Kudo, 2018; Meister, 2025). Given a string S, it finds the most probable sequence whose concatenation equals S, via the Viterbi algorithm (Viterbi, 1967). A subtokenizer $\mathcal{T}_s$ restricts this to $\mathcal{V}_s$ by assigning zero probability to tokens $v \notin \mathcal{V}_s$, while keeping original probabilities for $v \in \mathcal{V}_s$. No renormalization is needed, as rescaling does not affect the optimal Viterbi path. To build the vocabularies $\mathcal{V}_i$, we consider two main approaches.

Frequency-Based Extraction. Given a multilingual tokenizer $\mathcal{T}$, a language-specific vocabulary $\mathcal{V}_i$ of size k is built by retaining the k most frequent tokens on a corpus in $\mathcal{L}_i$. Even with k much smaller than $|\mathcal{V}|$, this approach preserves compression comparable to $\mathcal{T}$ (Appendix A.3.4, Tab 12), and the reduced vocabulary size directly translates into lower inference latency (Appendix A.6). However, the compression of the resulting subtokenizer is inherently bounded by that of $\mathcal{T}$.

Merging Monolingual Tokenizers. Monolingual tokenizers are trained on each corpus $\mathcal{D}_i$, and their vocabularies are merged into $\mathcal{V} = \bigcup \mathcal{V}_i$. Token probabilities are then re-estimated over the combined corpus $\mathcal{D} = \bigcup \mathcal{D}_i$ using the EM algorithm (Algorithm 1, Appendix A.1.1). Subtokenizers extracted for each $\mathcal{L}_i$ use the original $\mathcal{V}_i$, achieving compression rates comparable to independently trained monolingual tokenizers (Appendix A.3.3, Tab 11). As this approach achieves better compression and faster inference than frequency-based extraction, it is used as the default Unigram setting.

4.2 BPE Tokenizer

BPE builds an *ordered* sequence of merge rules by recursively combining frequent adjacent token pairs. This merge order dictates token priority during segmentation, tying the tokenizer to its merge table and making it non-trivial to combine independently trained monolingual BPE models. A common, albeit suboptimal, approach is to concatenate their merge tables by appending non-overlapping tokens from one vocabulary to the other. However,

early merges from one language can prevent tokens specific to the other from ever being formed, resulting in inefficiencies (Fig. 2).

Inspired by Purason et al. (2026), we propose a **sequential training** strategy (Algorithm 2). Given an ordered list of languages $\{\mathcal{L}_1, \ldots, \mathcal{L}_N\}$ with corpora $\{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$ and a per-language vocabulary budget k, it incrementally builds a multilingual tokenizer $\mathcal{T}$ while producing a monolingual subtokenizer $\mathcal{T}_i$ for each language. A BPE tokenizer of size k is first trained on $\mathcal{D}_1$, initializing the global tokenizer $\mathcal{T} = \mathcal{T}_1$. For each subsequent language $\mathcal{L}_i$, the corpus $\mathcal{D}_i$ is tokenized with $\mathcal{T}$, and a modified BPE procedure simultaneously trains $\mathcal{T}_i$ and extends $\mathcal{T}$. At each iteration, we identify the most frequent unigram u_m and bigram $b_m = (x, y)$ in $\mathcal{T}(\mathcal{D}_i)$, with respective counts f_u and f_b. If $f_u \geq f_b$, we add u_m to $\mathcal{T}_i$ along with all ancestor units and merges required to recompose it. Otherwise, since x and y are more frequent than (x, y) in $\mathcal{D}_i$ and thus already belong to $\mathcal{T}_i$, we add the merged unit (x, y) to both $\mathcal{T}_i$ and $\mathcal{T}$. As in standard BPE, occurrences of (x, y) are replaced with b_m in $\mathcal{T}(\mathcal{D}_i)$ and the frequencies are updated accordingly. This continues until $|\mathcal{V}_i| = k$. Crucially, the sequential merge order guarantees that every token in $\mathcal{T}_i$ can be formed (proof in A.1.2).

Tabs. 2 and 10 show that both $\mathcal{T}$ and each subtokenizer $\mathcal{T}_i$ achieve compression comparable to independently trained monolingual BPE models. Moreover, language ordering has negligible impact on compression (Tab. 7), so we use alphabetical order by default. For any subset of languages, a unified subtokenizer is formed by taking the union of their vocabularies and merges, preserving their original order in $\mathcal{T}$. Therefore, all tokens remain constructible (proof in A.1.2), unlike with naive concatenation, ensuring competitive compression across all selected languages.

5 Modular Tokenizers Resolve Multilingual Tokenizers' Inefficiencies

We now show that modular tokenizers resolve both inefficiencies documented in §3. As these hit low-resource languages hardest, we retrain, for this section only, monolingual, sequential, and joint BPE tokenizers on 21 languages where Estonian, Finnish, Lithuanian, Latvian, Slovak, and Swedish are replaced by Croatian, Nepali, Sinhala, Somali, Swahili, and Telugu (three new scripts; using FineWeb-2 (Penedo et al., 2025) data, same construction as before: 24k tokens per language, alphabetical order).

		NSL (↓)					
Model	Size	HR	NE	SI	SO	SW	TE
TinyAya	261k	1.27	1.41	9.08	1.61	1.39	1.68
NLLB	256k	1.24	1.30	1.50	1.29	1.26	1.40
Gemma3	262k	1.34	1.45	1.79	1.54	1.57	1.61
Llama3	128k	1.62	2.43	6.68	1.74	1.84	7.40
Qwen3	152k	1.66	4.15	5.38	1.74	1.84	6.33
Multi 21	384k	0.92	0.99	0.96	0.97	0.99	0.95
Seq. 24k	387k	0.95	0.97	0.95	0.96	0.98	0.97
Extracted 24k	24k	1.04	1.01	1.00	1.02	1.03	1.01

Table 2: NSL relative to 24k-token monolingual BPE references on low-resource languages. *Multi 21*: a standard BPE jointly trained on the same corpora; *Seq. 24k*: our sequential multilingual BPE (24k tokens per language); *Extracted 24k*: its per-language subtokenizers.

5.1 Multilingual Fairness

Our modular tokenizer yields subtokenizers on par with independently trained monolingual ones: every extracted 24k subtokenizer stays within 4% of its monolingual counterpart, on the low-resource languages (Tab. 2) as on the conservative set (Tab. 10). The full sequential tokenizer is equally fair (NSL 0.95–0.99), matching standard BPE trained on the same corpora at a comparable size — modularity costs nothing in compression, yet every language can be served by its own subtokenizer. The contrast is sharpest on the low-resource languages: Open-weight tokenizers instead inflate sequences even on languages they report as supported — up to $9.1\times$ for TinyAya on Sinhala (Salamanca et al., 2026). In absolute terms, ours cost a uniform 0.21–0.27 tokens per character across the six languages, against 0.31–0.48 for Gemma-3 and 0.29–2.44 for TinyAya (A.3.1): no language pays a disproportionate price for being under-represented.

5.2 Inference and Memory Efficiency

The fairness gains of §5.1 translate into efficiency: inference time and memory follow the two quantities a tokenizer controls — the active vocabulary size, whose impact fades with model size, and the sequence length it produces, whose impact persists. The vocabulary costs compute at the output layer: running with the full vocabulary (an illustrative extreme — inference would restrict to the target languages) costs $1.58\times$ the monolingual reference at 0.5B but $1.07\times$ at 30B (Tab. 3). It also costs mem-

Tokenizer	Lang	Normalized inference time (↓) 0.5B	2.5B	7B	30B
Gemma3	EN	1.49	1.26	1.12	1.05
	TE	2.46	2.09	1.84	1.73
Qwen3	EN	1.24	1.11	1.03	1.00
	TE	8.18	7.34	6.86	6.56
Seq 24k	EN	1.58	1.28	1.12	1.07
	TE	1.58	1.27	1.12	1.07
Extracted	(all langs)	0.99–1.04			

Table 3: Forward time on FLORES relative to a dedicated 24k monolingual tokenizer, across model scales. Full grid in App. F and architecture details in Tab. 23.

ory in its parameters: using a 24k sub-vocabulary instead of a conventional 256k removes 47%, 35%, and 10% of *all* parameters at 1B, 2.5B, and 30B for untied embeddings (31%, 21%, and 5% when tied). Compression, in contrast, is a property of the tokenizer alone: each token costs a forward pass and a KV-cache entry, so time and cache shrink with the sequence. In time, Gemma-3 runs $2.5\times$ slower than the monolingual reference on Telugu at 0.5B and still $1.7\times$ at 30B, while our extracted subtokenizers stay within 0.99–$1.04\times$ at every scale and language (full grid in Appendix F). In memory, Telugu sequences are $1.61\times$ the monolingual length under Gemma-3 vs. $1.01\times$ with our subtokenizer (Tab. 2), which thus needs 37% less cache, a cut over Gemma-3 of 22% (Croatian) to 44% (Sinhala), and $\sim$30% even on well-covered Bulgarian.

6 Training Multilingual Models with Modular Vocabularies

6.1 Experimental Settings

Training Data. For all 21 languages considered (Appendix A.2), we use CommonCrawl data processed as follows: text is extracted from HTML using `resiliparse`,[5] language-identified with `fastText`[6] to build a dedicated corpus for each language, deduplicated at the paragraph level via hashing, and quality-filtered using a `fastText` classifier trained to distinguish high-quality documents (Wikipedia, textbooks, news, scientific articles) from random web pages. We use a mixture of 70% English and 30% other languages for training.

Models. We use 2.5B non-embedding-parameter decoder-only Transformers (Vaswani et al., 2017) with a hidden dimension of 3072 (hyperparameters in Appendix B), trained on roughly 80B tokens with a context size of 4,096. Monolingual subtokenizers have a vocabulary budget of 24k tokens. As a baseline, standard training uses a single multilingual tokenizer with 384k tokens covering all languages. Merging monolingual Unigram tokenizers yields a vocabulary of 376,375 tokens, while sequential BPE yields 366,208 tokens. Since both approaches lead to the same conclusions (Tabs. 6 and 15), the merged Unigram tokenizer is our default modular tokenizer unless stated otherwise.

Evaluation. When using a subtokenizer, only the corresponding tokens are retained in the output matrix unless stated otherwise. For English, we evaluate on ARC Easy and Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), CommonSenseQA (Talmor et al., 2019), BeleBele (Bandarkar et al., 2024), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), and MMLU (Hendrycks et al., 2021), following the cloze form setting and normalization recommendations of Gu et al. (2025). For other languages, we use translated versions of ARC, HellaSwag, and MMLU from Thellmann et al. (2024), as well as BeleBele (Bandarkar et al., 2024). We refer to the average of these benchmarks as MCQ (Multi-Choice Questioning). For generation, we evaluate summarization on WikiLingua (Ladhak et al., 2020) in the monolingual setting, and translation on FLORES-200 (Costa-jussà et al., 2024) (English to other languages) in the multilingual setting. All evaluations are 5-shot, except HellaSwag and BeleBele which are 0-shot.

6.2 Preliminary Analysis

For simplicity, our experiments use the combination of all subtokenizers (referred to as *full tokenizer* $\mathcal{T}$) as a reference to analyze model behavior when multiple subtokenizers are merged, even though in practice, only a subset would typically be used. Additional experiments with alternative subtokenizer combinations are presented in C.1.

To study the effects of training with different tokenizers, we train a model referred to as *Mono-only* on monolingual batches, where each language $\mathcal{L}_i$ is tokenized with $\mathcal{T}_i$ and the loss is computed only over $\mathcal{V}_i$ via the corresponding output layer activations. The baseline follows the same *monolingual batching scheme* but with a single shared multilingual tokenizer. *Mono-only* matches the baseline on monolingual tasks including MCQ (Tab. 6)

[5] https://resiliparse.chatnoir.eu
[6] https://fasttext.cc

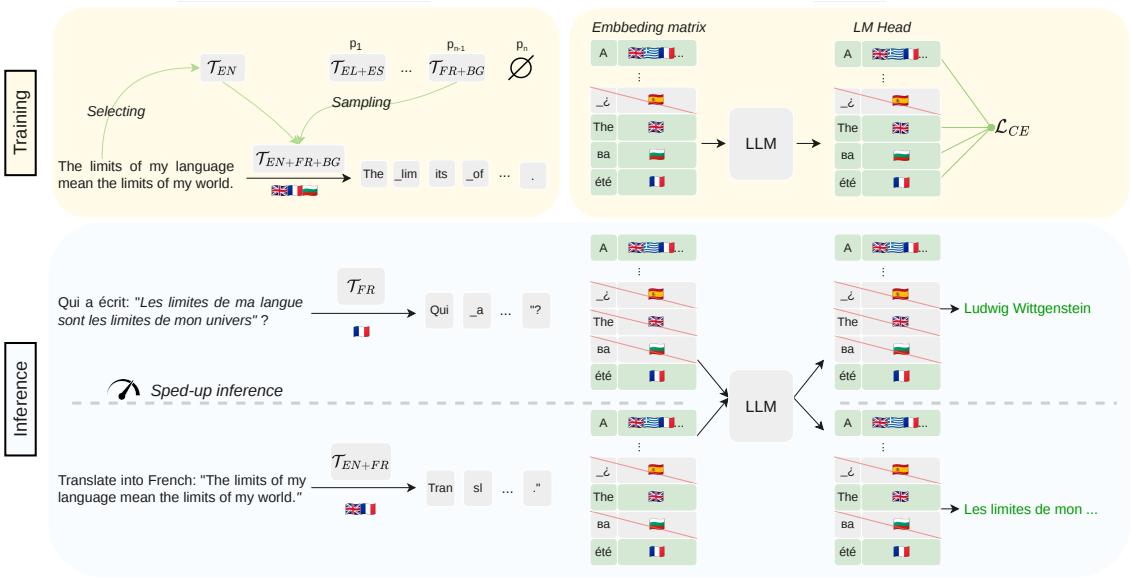


Figure 3: Overview of our multilingual training method and modular inference. **Top**: At training, monolingual batches are formed using either strictly monolingual or combined with n other languages (here, $n = 2$), restricting logit computation to that specific sub-vocabulary. **Bottom**: At inference, applying task-specific tokenizers (e.g., French-only for QA in French or English+French for translation tasks) shrinks the active vocabulary. This specialized approach reduces both the output token count and FLOPs, improving inference efficiency.

Model	Infer.	CS	EN	FR	IT	NL	PT
Baseline	Full	11.9	19.2	15.3	13.4	18.0	17.9
Mono-only	Lang	12.2	**21.8**	14.9	13.6	**19.5**	**19.4**
	Full	4.2	20.4	7.2	3.8	4.6	5.1
	Full - Lang	11.9	20.9	14.1	12.9	18.1	18.5
$Samp_{\text{Full}}$	Lang	12.6	19.3	14.8	14.2	17.8	17.8
	Full	12.9	20.1	13.6	**14.4**	18.2	17.2
$Samp_{\text{Sub 2}}$	Lang	12.3	20.1	**16.1**	13.2	18.9	18.5
	Full	**13.0**	20.2	15.9	12.7	18.8	17.5

Table 4: Summarization across pretraining setups (Unigram). *Infer.* is the inference tokenizer. *Lang* is the monolingual tokenizer for the target language; *Full* is the full tokenizer. *Full-Lang*: inputs tokenized with *Full*, predictions restricted to *Lang* vocabulary.

	BG	DA	EL	FR	IT
Mono-only (EN+Lang)	1.0	2.8	5.3	15.0	6.9
Mono-only (EN|Lang - Lang)	1.6	1.5	12.3	26.0	12.7
Baseline (Full)	**26.1**	**35.3**	**17.1**	**32.4**	**20.1**

Table 5: Translation results with the settings of Tab. 4. *EN+Lang*: inference with the union of English and Lang subtokenizers. *EN|Lang*: few-shot examples concatenate source tokenized with English subtokenizer and target with Lang subtokenizer; prediction restricted to Lang vocabulary.

and summarization (Tab. 4), when each language is evaluated with its corresponding subtokenizer. However, combining subtokenizers at inference degrades performance, particularly for non-English generation, where outputs often mix tokens from different languages. This is expected: since token prediction is restricted to monolingual vocabularies during training, merging them at inference introduces a distribution shift in the prediction space. To verify this, we evaluate the *Full-Lang* setting on summarization (Tab. 4), where inputs are tokenized with $\mathcal{T}$ but generation is constrained to $\mathcal{V}_i$. Recovering baseline performance confirms that the distribution shift is the cause of degradation. English is less affected, as its dominance in training naturally biases *Mono-only* toward English tokens.

This bias toward English is more pronounced when languages are mixed in context. We evaluate this in a 5-shot translation setting using the combination of the English and target language subtokenizers (*EN+Lang* in Tab. 5). Here, performance drops substantially compared to the baseline. The distribution shift in the prediction space partly explains this degradation, but does not fully account for it. To investigate further, we modify the evaluation protocol: source sentences are tokenized with the English subtokenizer and target translations with the target-language one, concatenated for each few-shot example, with generation restricted to the target vocabulary (*EN* | *Lang* in

Model	Infer.	EN	BG		DA		EL		FR		IT		SK	
		MCQ	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT
Baseline	Full	52.9	37.3	26.1	40.4	**35.3**	37.3	17.1	39.5	32.4	38.8	20.1	38.1	21.0
Mono-only	Lang	52.6	38.3	1.0	40.5	2.8	37.9	5.3	40.3	15.0	38.9	6.9	38.9	1.6
	Full	49.6	36.3	0.9	38.3	2.4	37.1	2.5	37.8	7.4	37.2	3.0	36.1	1.4
$Samp_{\text{Full}}$	Lang	**53.2**	38.1	**27.9**	**41.6**	35.1	**38.0**	16.9	39.3	32.5	**39.1**	19.8	38.3	20.3
	Full	53.0	38.0	27.7	41.5	35.2	37.9	16.8	39.4	**32.6**	38.8	**20.2**	38.2	**21.1**
$Samp_{\text{Sub}\,2}$	Lang	52.5	**38.9**	27.1	40.8	35.0	37.8	**17.5**	**40.4**	32.2	38.7	19.8	**39.3**	20.2
	Full	52.1	38.8	26.3	40.2	34.7	37.4	17.3	39.3	32.1	37.6	19.7	37.3	19.8

Table 6: Multiple-Choice Questioning (MCQ) and Machine Translation (MT) performance under different pretraining sampling schemes for Unigram Tokenizers. *Infer.* denotes the tokenizer used at inference; *Full* corresponds to the full tokenizer; *Lang* to the monolingual tokenizer for MCQ and the combined English–target tokenizer for MT.

Tab. 5). This mitigates the distribution shift and improves performance for some languages, but results remain far from the baseline. Despite these constraints, the model frequently produces English or mixed-language outputs, even for languages such as Bulgarian, which uses the Cyrillic script and shares few tokens with English. This suggests the issue goes beyond the prediction space distribution shift: during training, each sequence contains tokens from a single language only, so sequences mixing tokens from multiple languages at inference create out-of-distribution patterns that hinder proper language selection during translation.

6.3 Training with Sampled Subtokenizers

To mitigate this, we design $Samp_{\text{Full}}$, a middle-ground training regime between using a single large multilingual tokenizer and monolingual subtokenizers. The former enables proper language selection but is costly; the latter is efficient but tends to produce language mixtures at inference. For a batch in language $\mathcal{L}_i$, we tokenize using either $\mathcal{T}$ or $\mathcal{T}_i$. Using $\mathcal{T}$ introduces a small fraction of tokens from other languages (0.5–5%, depending on the language; Appendix D.1). We hypothesize that exposing the model to such mixed-token sequences acts as regularization, improving its ability to generate coherent text in $\mathcal{L}_i$ even when foreign tokens appear in context. As a result, the model assigns more probability mass to target-language tokens during translation, enabling proper language switching and recovering baseline performance.

As the number of languages grows, the *full tokenizer* becomes very large (Appendix A.5), making training expensive and requiring approximate softmax methods (Joulin et al., 2017; Baevski and Auli, 2019). To improve scalability, we propose $Samp_{\text{Sub}\,n}$ (Fig. 3): text in language $\mathcal{L}_i$ is tokenized either with $\mathcal{T}_i$ alone or combined with n additional subtokenizers (instead of $\mathcal{T}$ as in $Samp_{\text{Full}}$ method). We decided to sample the additional subtokenizers proportionally to the frequencies of their corresponding languages in the training data. This sampling strategy ensures two key properties that we found to be important to match the baseline performance. First, it calibrates token probabilities across languages: tokens from overrepresented languages tend to have larger logits, so including their subtokenizers more often as "negatives" mitigates this bias. Second, when tokenizing texts from different languages with $\mathcal{T}$, English contributes the largest share of foreign tokens due to its prevalence in training data (Appendix D.1). So subtokenizers introducing the most out-of-language tokens should be selected more frequently to reproduce the regularizing effect of the $Samp_{\text{Full}}$ method.

To simplify training, we retain the *monolingual batching scheme*: each batch uses a sampled subtokenizer (either $\mathcal{T}_i$ or a combination), ensuring all tokens belong to the same vocabulary subset and allowing the loss to be computed over that subset only. This avoids the training overhead caused by near-linear vocabulary growth, without requiring approximate softmax methods. Unless stated otherwise, we sample the monolingual tokenizer 50% of the time for $Samp_{\text{Full}}$ and $Samp_{\text{Sub}\,n}$. These strategies recover baseline performance across MCQ, translation (Tab. 6), and summarization (Tab. 4). Crucially, $Samp_{\text{Sub}\,n}$ maintains strong performance when using the full tokenizer at inference despite never seeing it during training, demonstrating that the model generalizes to any combination of subto-

n lang.	1	2	3	4
Seq.	$1.01_{\pm.01}$	$1.00_{\pm.01}$	$0.99_{\pm.01}$	$0.99_{\pm.01}$
Concat.	$1.00_{\pm.00}$	$1.22_{\pm.27}$	$1.30_{\pm.27}$	$1.33_{\pm.25}$

Table 7: Mean NSL $\pm$ std across languages for n-language subtokenizers: concatenated monolingual tokenizers vs. extraction from sequential BPEs trained under different language orders (details in A.4.2).

kenizers without sacrificing performance.

6.4 Memory Footprint at Many Languages

Scaling to more languages grows the vocabulary near-linearly (Appendix A.5), and with it the memory-resident vocabulary state: embeddings, output matrix, optimizer states. CPU offloading lifts this bottleneck by keeping this state in CPU RAM (Miao et al., 2021; Ren et al., 2021), each step touching only the sampled subtokenizers' rows. Updates are pipelined following the delayed scheme of Ren et al. (2021): while the GPU computes step t, the CPU updates the rows active at step $t-1$ with their gradients; the transfers (gradients of $t-1$ to the CPU, rows for step $t+1$ to the GPU) run asynchronously from pinned buffers and hide behind compute as well. Every per-step cost therefore depends only on the languages sampled, not on the total vocabulary: we measure identical step times whether CPU RAM holds 21, 100, or 250 languages. To introduce no overhead, this off-GPU work must complete within one GPU step, which we verify on a naive prototype (Appendix E), separate from our training runs: at batch 128, on $\mathit{Samp}_{\mathrm{Sub}\,5}$ batches, the full off-GPU work takes 1.2 s vs. a 1.5 s GPU step at 1B, 2.6 s vs. 3.1 s at 2.5B, and 4.7 s vs. 7.4 s at 7B, and monolingual batches stay far below. These timings can be further reduced using the fused optimizer of Ren et al. (2021), which significantly speeds up the CPU update that dominates the off-GPU work.

7 Ablations

Sequential BPE is order-robust and preserves compression. Tab. 7 compares subtokenizers covering n languages built via (i) sequential BPE training followed by subtokenizer extraction and merging as in § 4.2, evaluated across multiple training orders, and (ii) concatenation of independently trained monolingual tokenizers. For $n = 1$, subtokenizers extracted from sequential BPE achieve compression comparable to independently trained monolingual tokenizers. For larger n, concatenation becomes sensitive to both language selection and order, leading to unstable and degraded compression. In contrast, sequential BPE is insensitive to both training and combination orders: fixed merge priorities avoid ordering ambiguity, preserving strong and consistent compression across all languages in the considered set.

Model	BG	DA	EL	FR	IT	SK
Baseline	26.1	35.3	17.1	32.4	20.1	21.0
$\mathit{Samp}_{\mathrm{Sub}\,1}$	21.4	26.4	14.7	27.4	16.7	13.6
$\mathit{Samp}_{\mathrm{Sub}\,2}$	26.3	34.7	**17.3**	32.1	19.7	19.8
$\mathit{Samp}_{\mathrm{Sub}\,5}$	**28.0**	**36.1**	17.2	32.4	**20.7**	**21.2**
$\mathit{Samp}_{\mathrm{Full}}$	27.7	35.2	16.8	**32.6**	20.2	21.1

Table 8: Translation performance when varing n in $\mathit{Samp}_{\mathrm{Sub}\,n}$ using the Full tokenizer (Unigram).

Sampling more subtokenizers improves performance. $\mathit{Samp}_{\mathrm{Sub}\,n}$ interpolates between *Mono-only* $\sim\mathit{Samp}_{\mathrm{Sub}\,0}$ and $\mathit{Samp}_{\mathrm{Full}}$ $\sim\mathit{Samp}_{\mathrm{Sub}\,n-1}$ for n languages. Increasing n improves the model's ability to handle mixed-language contexts, as in translation (Tab. 8). Even small n achieves near-baseline performance, while larger n is beneficial when scaling to more languages. The main constraint is memory, as larger n increases the output matrix size. Since no performance degradation was observed, n can be as large as memory permits.

8 Conclusions

In this work, we highlight a key limitation of multilingual LLMs: their reliance on a single large shared tokenizer causes uneven compression across languages and memory and compute inefficiencies, especially when models are deployed on a subset of the supported languages. To address this, we introduce modular multilingual tokenizers for both BPE and Unigram, decomposable into fair and tailorable subtokenizers. We further propose a pretraining approach that samples subtokenizers per training batch, computing logits only over the relevant vocabulary subset, curbing the cost of large vocabularies. This enables flexible subtokenizer selection at inference. Our results show that this yields more efficient, fair, and modular multilingual LLMs — better cross-lingual balance, lower memory usage, and faster inference than shared-tokenizer models — at competitive performance.

Limitations

While our approach improves compression fairness and inference efficiency, some limitations remain. First, our low-resource evaluation is tokenizer-level: compression and fairness are measured on six genuinely low-resource languages, but reliable downstream validation requires stronger pretraining for these languages — our preliminary runs were too noisy to be conclusive — and remains future work. Second, subtokenizer selection at inference requires knowing the target language(s) in advance; language identification or a broader subtokenizer union can cover uncertain inputs, but we do not evaluate such pipelines.

Acknowledgments

François Yvon has been partly funded by the French National Funding Agency (ANR) under the France 2030 program (ref. ANR-23-IACL-0007).

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A. Smith. 2024. MAGNET: Improving the multilingual fairness of language models with adaptive gradient-based tokenization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. 2023. Do all languages cost the same? tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore. Association for Computational Linguistics.

Catherine Arnett, Tyler A. Chang, Stella Biderman, and Ben Bergen. 2025. Explaining and mitigating crosslingual tokenizer inequities. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*.

Lucas Bandarkar, Davis Liang, Benjamin Muller, Mikel Artetxe, Satya Narayan Shukla, Donald Husa, Naman Goyal, Abhinandan Krishnan, Luke Zettlemoyer, and Madian Khabsa. 2024. The belebele benchmark: a parallel reading comprehension dataset in 122 language variants. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 749–775, Bangkok, Thailand. Association for Computational Linguistics.

Martin Berglund and Brink van der Merwe. 2023. Formalizing BPE tokenization. *Electronic Proceedings in Theoretical Computer Science*, 388:16–27.

Yonatan Bisk, Rowan Zellers, Ronan bras, Jianfeng Gao, and Choi Yejin. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:7432–7439.

Daiki Chijiwa, Taku Hasegawa, Kyosuke Nishida, Shin'ya Yamaguchi, Tomoya Ohba, Tamao Sakao, and Susumu Takeuchi. 2026. Lossless vocabulary reduction for auto-regressive language models. In *The Fourteenth International Conference on Learning Representations*.

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.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *Preprint*, arXiv:1803.05457.

Marta R. Costa-jussà, 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, and 20 others. 2024. Scaling neural machine translation to 200 languages. *Nature*, 630(8018):841–846. ISBN: 1476-4687 tex.date-added: 2024-08-21 15:32:57 +0200 tex.date-modified: 2024-08-21 15:32:57 +0200.

Zoltan Csaki, Bo Li, Jonathan Lingjie Li, Qiantong Xu, Pian Pawakapan, Leon Zhang, Yun Du, Hengyu Zhao, Changran Hu, and Urmish Thakker. 2024. SambaLingo: Teaching large language models new languages. In *Proceedings of the Fourth Workshop on Multilingual Representation Learning (MRL 2024)*, pages 1–21, Miami, Florida, USA. Association for Computational Linguistics.

Yiming Cui, Ziqing Yang, and Xin Yao. 2024. Efficient and effective text encoding for chinese llama and alpaca. *Preprint*, arXiv:2304.08177.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. 2024. Getting the most out of your tokenizer for pre-training and domain adaptation. In *Forty-first International Conference on Machine Learning*.

Konstantin Dobler and Gerard de Melo. 2023. FOCUS: Effective embedding initialization for monolingual specialization of multilingual models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13440–13454,

Singapore. Association for Computational Linguistics.

Negar Foroutan, Clara Meister, Debjit Paul, Joel Niklaus, Sina Ahmadi, Antoine Bosselut, and Rico Sennrich. 2026. Parity-aware byte-pair encoding: Improving cross-lingual fairness in tokenization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7514–7538, San Diego, California, United States. Association for Computational Linguistics.

Kazuki Fujii, Taishi Nakamura, Mengsay Loem, Hiroki Iida, Masanari Ohi, Kakeru Hattori, Hirai Shota, Sakae Mizuki, Rio Yokota, and Naoaki Okazaki. 2024. Continual pre-training for cross-lingual LLM adaptation: Enhancing japanese language capabilities. In *First Conference on Language Modeling*, COLM 2024.

Philip Gage. 1994. A New Algorithm for Data Compression. *Computer Users Journal*, 12(2):23–38.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, and Paolo Torroni. 2022. Fast vocabulary transfer for language model compression. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 409–416, Abu Dhabi, UAE. Association for Computational Linguistics.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2274–2286, Bangkok, Thailand. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. 2025. OLMES: A standard for language model evaluations. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5020–5048, Albuquerque, New Mexico. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Kenji Imamura, Masao Utiyama, and Eiichiro Sumita. 2023. Pivot translation for zero-resource language pairs based on a multilingual pretrained model. In *Proceedings of Machine Translation Summit XIX, Vol. 1: Research Track*, pages 348–359, Macau SAR, China. Asia-Pacific Association for Machine Translation.

Armand Joulin, Moustapha Cissé, David Grangier, Hervé Jégou, and 1 others. 2017. Efficient softmax approximation for gpus. In *International conference on machine learning*, pages 1302–1310. PMLR.

Artur Kiulian, Anton Polishko, Mykola Khandoga, Yevhen Kostiuk, Guillermo Gabrielli, Łukasz Gagała, Fadi Zaraket, Qusai Abu Obaida, Hrishikesh Garud, Wendy Wing Yee Mak, Dmytro Chaplynskyi, Selma Amor, and Grigol Peradze. 2025. From English-centric to effective bilingual: LLMs with custom tokenizers for underrepresented languages. In *Proceedings of the Fourth Ukrainian Natural Language Processing Workshop (UNLP 2025)*, pages 1–13, Vienna, Austria (online). 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.

Faisal Ladhak, Esin Durmus, Claire Cardie, and Kathleen McKeown. 2020. WikiLingua: A new benchmark dataset for cross-lingual abstractive summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4034–4048, Online. Association for Computational Linguistics.

Davis Liang, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. 2023. XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13142–13152, Singapore. Association for Computational Linguistics.

Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. 2024. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15059–15076, Bangkok, Thailand. Association for Computational Linguistics.

Peiqin Lin, Shaoxiong Ji, Jörg Tiedemann, André F. T. Martins, and Hinrich Schütze. 2024. Mala-500: Massive language adaptation of large language models. *Preprint*, arXiv:2401.13303.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A. Smith, and Yejin Choi. 2025. SuperBPE: Space travel for language models. In *Second Conference on Language Modeling*.

Clara Meister. 2025. UnigramLM: An Attempt at Writing The Missing Manual. Blog post.

Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2021. Het: scaling out huge embedding model training via cache-enabled distributed framework. *Proceedings of the VLDB Endowment*, 15(2):312–320.

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. 2022. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3992–4006, Seattle, United States. Association for Computational Linguistics.

Itay Nakash, Nitay Calderon, Eyal Ben-David, Elad Hoffer, and Roi Reichart. 2025. AdaptiVocab: Enhancing LLM efficiency in focused domains through lightweight vocabulary adaptation. In *Second Conference on Language Modeling*, COLM 2025.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. Fineweb2: One pipeline to scale them all — adapting pre-training data processing to every language. In *Second Conference on Language Modeling*.

Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Vijini Pilana Liyanage and François Yvon. 2026. AdaptBPE: From general purpose to specialized tokenizers. In *Proceedings of the 19th conference of the European chapter of the Association for Computational Linguistics (volume 1: Long papers)*, pages 2607–2620, Rabat, Morocco. Association for Computational Linguistics.

Taido Purason, Pavel Chizhov, Ivan P. Yamshchikov, and Mark Fishel. 2026. Teaching old tokenizers new words: Efficient tokenizer adaptation for pretrained models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 6492–6516, Rabat, Morocco. Association for Computational Linguistics.

Lawrence Rabiner and Biing Hwang Juang. 1993. Fundamentals of speech recognition.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. Zero-offload: Democratizing billion-scale model training. *Preprint*, arXiv:2101.06840.

Alejandro R. Salamanca, Diana Abagyan, Daniel D'souza, Ammar Khairi, David Mora, Saurabh Dash, Viraat Aryabumi, Sara Rajaee, Mehrnaz Mofakhami, Ananya Sahu, Thomas Euyang, Brittawnya Prince, Madeline Smith, Hangyu Lin, Acyr Locatelli, Sara Hooker, Tom Kocmi, Aidan Gomez, Ivan Zhang, and 7 others. 2026. Tiny Aya: Bridging Scale and Multilingual Depth. *Preprint*, arXiv:2603.11510.

David Samuel, Vladislav Mikhailov, Erik Velldal, Lilja Øvrelid, Lucas Georges Gabriel Charpentier, Andrey Kutuzov, and Stephan Oepen. 2025. Small languages, big models: A study of continual training on languages of Norway. In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pages 573–608, Tallinn, Estonia. University of Tartu Library.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Craig W Schmidt, Varshini Reddy, Chris Tanner, and Yuval Pinter. 2025. Boundless byte pair encoding: Breaking the pre-tokenization barrier. In *Second Conference on Language Modeling*.

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.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages

4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Atula Tejaswi, Nilesh Gupta, and Eunsol Choi. 2024. Exploring design choices for building language-specific LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10485–10500, Miami, Florida, USA. Association for Computational Linguistics.

Klaudia Thellmann, Bernhard Stadler, Michael Fromm, Jasper Schulze Buschhoff, Alex Jude, Fabio Barth, Johannes Leveling, Nicolas Flores-Herr, Joachim Köhler, René Jäkel, and Mehdi Ali. 2024. Towards Multilingual LLM Evaluation for European Languages. *Preprint*, arXiv:2410.08928.

Asahi Ushio, Yi Zhou, and Jose Camacho-Collados. 2023. Efficient multilingual language model compression through vocabulary trimming. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14725–14739, Singapore. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Miles Williams and Nikolaos Aletras. 2025. Vocabulary-level memory efficiency for language model fine-tuning. In *Proceedings of the 10th Workshop on Representation Learning for NLP (RepL4NLP-2025)*, pages 185–196, Albuquerque, NM. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Hanling Zhang, Yayu Zhou, Tongcheng Fang, Zhihang Yuan, Guohao Dai, Wanli Ouyang, and Yu Wang. 2026. VocabTailor: Dynamic vocabulary selection for downstream tasks in small language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 28453–28471, San Diego, California, United States. Association for Computational Linguistics.

Bo Zheng, Li Dong, Shaohan Huang, Saksham Singhal, Wanxiang Che, Ting Liu, Xia Song, and Furu Wei. 2021. Allocating large vocabulary capacity for cross-lingual language model pre-training. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3203–3215, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. 2023. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada. Association for Computational Linguistics.

A Tokenizer Study

A.1 Tokenizer construction

A.1.1 Multilingual Unigram Tokenizer

Forward–Backward Re-estimation over Merged Vocabularies. To establish a unified probability space for the merged vocabulary $V = \bigcup V_i$, parameter estimation is performed over the aggregate multilingual corpus $\mathcal{D} = \bigcup \mathcal{D}_i$. As detailed in Algorithm 1, the Expectation-Maximization (EM) procedure reestimates token probabilities $p(v)$ using the forward-backward algorithm (Rabiner and Juang, 1993).

For a sentence $S \in \mathcal{D}$ of length n, the forward and backward variables, $\alpha(i)$ and $\beta(i)$, are computed to represent the marginal probabilities of all segmentations for the prefix $S[0 : i]$ and suffix $S[i : n]$, respectively.

Under the Unigram independence assumption, the total sentence likelihood is given by $P(S) = \alpha(n) = \beta(0)$. The E-step utilizes these variables to calculate the expected counts $E[c(v)]$ for each token $v \in V$ by marginalizing over all valid segmentation paths in the corpus. In the subsequent M-step, the parameters are updated via the normalization:

$$p(v) = \frac{E[c(v)]}{\sum_{v' \in V} E[c(v')]}$$

This iterative process ensures that the probabilities for the merged vocabulary are normalized and remain consistent with the multilingual data distribution.

A.1.2 Multilingual BPE Tokenizer

Each BPE tokenizer $\mathcal{T}$ consists of an ordered list of merges $\mathcal{T}$.merges and of tokens $\mathcal{T}$.vocab.

Proof of Properness. To prove the correctness of Algorithm 2, we show that it preserves the *properness* of the merge sequence (Berglund and van der Merwe, 2023). This property ensures that any compound token is introduced only after all tokens required to construct it have already been included.

Formally, let $\mu = [\mu_0, \mu_1, \ldots, \mu_N]$ be the sequence of merges, where each $\mu_i = (x_i, y_i) \to z_i$. The sequence μ is *proper* if for all i:

- $x_i \in \Sigma$ or $\exists j < i$ such that $x_i = z_j$,
- $y_i \in \Sigma$ or $\exists j < i$ such that $y_i = z_j$.

That is, both constituents of any merge must either belong to the base alphabet Σ or have been produced by a previous merge. Note that any prefix of a proper merge sequence is also proper.

Global tokenizer $\mathcal{T}$: The global tokenizer $\mathcal{T}$ is initialized using standard BPE on $\mathcal{D}_1$, which is known to produce a proper merge sequence. At each subsequent step, Algorithm 2 only appends a merge (x, y) to $\mathcal{T}$ when both x and y already belong to $\mathcal{T}$. Since (x, y) is selected from $\mathcal{T}(\mathcal{D}_i)$, both x and y are already representable using $\mathcal{T}$. Therefore, any newly added merge satisfies the properness condition, and $\mathcal{T}$ remains proper by closure under extension.

Language-specific tokenizers $\mathcal{T}_i$: Each $\mathcal{T}_i$ is constructed by adding tokens together with all their ancestors in $\mathcal{T}$, in the order induced by $\mathcal{T}$. Since $\mathcal{T}$ is proper, every merge depends only on tokens that appear earlier in this order. Thus, when a merge $(x, y) \to z$ is included in $\mathcal{M}_i$, both x and y have already been added to $\mathcal{V}_i$. Moreover, the merge sequence of $\mathcal{T}_i$ is an order-preserving subsequence of that of $\mathcal{T}$ that is closed under ancestry up to the vocabulary budget k. Although some tokens may be omitted due to truncation, no merge is included without its dependencies. Therefore, the merge sequence of $\mathcal{T}_i$ is proper.

Conclusion. Both the global tokenizer $\mathcal{T}$ and each $\mathcal{T}_i$ maintain proper merge sequences, ensuring that all constructed tokenizers are valid BPE tokenizers.

Proof of Properness of Combined Subtokenizers. For any subset of languages $\{\mathcal{L}_1, \ldots, \mathcal{L}_k\}$, let $\mathcal{T}_{\text{comb}}$ be the tokenizer obtained by taking the union of their vocabularies and merges and reordering them according to the global tokenizer $\mathcal{T}$. Since each language-specific tokenizer $\mathcal{T}_i$ is proper, every token it contains is associated with a valid ancestral merge chain in the global sequence $\mathcal{T}$. As $\mathcal{T}_{\text{comb}}$ is formed by taking the union of such tokens and preserving their global ordering, it remains closed under these ancestral dependencies. Therefore, every token in $\mathcal{T}_{\text{comb}}$ is constructible from Σ, and $\mathcal{T}_{\text{comb}}$ is proper.

In contrast, arbitrary concatenation of independently trained tokenizers may introduce tokens whose merge histories are incompatible or incomplete with respect to $\mathcal{T}$, leading to tokens that are not jointly constructible within a shared merge graph.

Algorithm 1 Unigram Probabilities Estimation via Expectation-Maximization

Require: Multilingual Corpus $\mathcal{D}$, Merged Vocabulary V

Initialization: Set $p(v) = \frac{1}{|V|}$ for all $v \in V$

repeat
  $E[c(v)] \leftarrow 0, \quad \forall v \in V$ ▷ Reset expected counts

  for all sentences $S \in \mathcal{D}$ **do**
    $n \leftarrow |S|$
    $\alpha(0) \leftarrow 1, \quad \beta(n) \leftarrow 1$

    for $j = 1$ **to** n **do** ▷ E-step: Forward Pass
      $\alpha(j) \leftarrow \sum_{i:0 \leq i < j, S[i:j] \in V} \alpha(i) \cdot p(S[i:j])$
    end for

    for $i = n - 1$ **down to** 0 **do** ▷ E-step: Backward Pass
      $\beta(i) \leftarrow \sum_{j:i < j \leq n, S[i:j] \in V} p(S[i:j]) \cdot \beta(j)$
    end for

    for $i = 0$ **to** $n - 1$ **do** ▷ E-step: Expectation Accumulation
      for $j = i + 1$ **to** n **do**
        if $S[i:j] \in V$ **then**
          $v \leftarrow S[i:j]$
          $E[c(v)] \leftarrow E[c(v)] + \frac{\alpha(i) \cdot p(v) \cdot \beta(j)}{\alpha(n)}$
        end if
      end for
    end for

  end for

  $Z \leftarrow \sum_{v' \in V} E[c(v')]$ ▷ M-step: Probabilities Update
  for all $v \in V$ **do**
    $p(v) \leftarrow \frac{E[c(v)]}{Z}$
  end for

until Log-likelihood $\sum_{S \in \mathcal{D}} \log \alpha(n)$ converges

Algorithm 2 Sequential Multilingual BPE with Language-Specific Extraction

Inputs: Ordered languages $\{\mathcal{L}_1, \ldots, \mathcal{L}_N\}$; corresponding corpora $\{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$; vocabulary budget k per language
Outputs: Global multilingual tokenizer $\mathcal{T}$; language-specific tokenizers $\{\mathcal{T}_i\}_{i=1}^{N}$, each preserving the merge order of $\mathcal{T}$ and containing at most k tokens

$\mathcal{T}_1 \leftarrow$ Train BPE with $|\mathcal{V}_1| = k$ on $\mathcal{D}_1$
$\mathcal{T} \leftarrow \mathcal{T}_1$

for $i = 2$ to N **do**
    $\mathcal{D}_i^{tok} \leftarrow \mathcal{T}(\mathcal{D}_i)$
    Compute unigram and bigram frequencies in $\mathcal{D}_i^{tok}$
    Initialize $\mathcal{V}_i \leftarrow \emptyset, \quad \mathcal{M}_i \leftarrow \emptyset$

    while $|\mathcal{V}_i| < k$ **do**
        $u_m \leftarrow$ most frequent unigram with frequency f_u
        $b_m = (x, y) \leftarrow$ most frequent bigram with frequency f_b
        if $f_u \geq f_b$ **then**
            $t_{\text{new}} \leftarrow u_m$
        else
            $t_{\text{new}} \leftarrow \textsc{Merge}(x, y)$
            Replace all occurrences of (x, y) with t_{new} in $\mathcal{D}_i^{tok}$
            Update unigram and bigram frequencies
            Append (x, y) to $\mathcal{T}$.merges
            Append t_{new} to $\mathcal{T}$.vocab
        end if

        $(\mathcal{A}, \mathcal{P}) \leftarrow$ ancestors of t_{new} in $\mathcal{T}$ and corresponding merges (ordered as in $\mathcal{T}$)
        for $(t, merge) \in (\mathcal{A}, \mathcal{P})$ **do**
            if $|\mathcal{V}_i| < k$ **then**
                $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \{t\}$
                if $merge \neq$ None **then** ▷ None for original bytes/characters
                    $\mathcal{M}_i \leftarrow \mathcal{M}_i \cup \{merge\}$
                end if
            else
                break
            end if
        end for

    end while
    Order $\mathcal{V}_i$ and $\mathcal{M}_i$ according to $\mathcal{T}$
    Define $\mathcal{T}_i$ from $(\mathcal{V}_i, \mathcal{M}_i)$
end for
return $\mathcal{T}$, $\{\mathcal{T}_i\}_{i=1}^{N}$

Model	BG	CS	DA	DE	EL	EN	ES	FR	HR	HU	IT	NE	NL	PL	PT	RO	SI	SL	SO	SW	TE
TinyAya	0.32	0.29	0.27	0.23	0.31	**0.21**	0.23	0.24	0.30	0.31	0.23	0.33	0.25	0.29	0.24	0.28	2.44	0.30	0.37	0.29	0.39
NLLB	0.30	0.32	0.26	0.27	0.33	0.25	0.25	0.28	0.29	0.29	0.26	0.30	0.26	0.31	0.26	0.29	0.40	0.29	0.29	0.26	0.32
Gemma3	0.33	0.33	0.29	0.24	0.38	**0.21**	0.23	0.25	0.31	0.33	0.25	0.34	0.26	0.30	0.25	0.29	0.48	0.33	0.35	0.33	0.37
Llama3	0.41	0.33	0.33	0.28	0.39	**0.21**	0.27	0.28	0.38	0.42	0.29	0.56	0.30	0.37	0.29	0.35	1.79	0.38	0.40	0.39	1.70
Qwen3	0.44	0.45	0.33	0.28	0.86	**0.21**	0.27	0.27	0.39	0.42	0.29	0.96	0.30	0.35	0.28	0.35	1.45	0.39	0.39	0.39	1.46
BPE mono 24k	**0.23**	0.24	0.23	**0.21**	**0.22**	0.22	0.22	0.23	0.23	0.23	**0.21**	**0.23**	**0.21**	**0.22**	0.22	0.23	0.27	0.23	0.23	**0.21**	0.23
Extracted 24k	**0.23**	0.24	0.23	0.22	**0.22**	0.22	0.22	0.23	0.24	0.24	0.22	**0.23**	0.22	0.23	0.22	0.23	0.27	0.24	0.23	0.22	0.23
Seq. 24k	**0.23**	**0.23**	**0.22**	**0.21**	**0.22**	**0.21**	**0.21**	**0.22**	**0.22**	**0.22**	**0.21**	**0.23**	**0.21**	**0.22**	**0.21**	**0.22**	**0.26**	**0.22**	**0.22**	**0.21**	**0.22**

Table 9: **Tokens per character** (↓) on FLORES-200; lower and more uniform is fairer.

A.2 Language choice

The selected dataset comprises twenty-one official working languages of the European Union, strategically chosen to provide a rigorous and representative linguistic testbed. We exclude Croatian, Irish and Maltese because of the poor coverage of these languages in our training data. A primary motivation for utilizing this specific European cohort is its coverage of three distinct scripts: Latin, Cyrillic, and Greek. By evaluating our method across these diverse writing systems, we can effectively demonstrate the framework's modularity and robust handling of script-level variation. The chosen languages are named using the ISO 639-1 language codes:[7]

- **Latin Script (19 languages)**: Czech (cs), Danish (da), German (de), English (en), Spanish (es), Estonian (et), Finnish (fi), French (fr), Hungarian (hu), Italian (it), Lithuanian (lt), Latvian (lv), Dutch (nl), Polish (pl), Portuguese (pt), Romanian (ro), Slovak (sk), Slovenian (sl), Swedish (sv);
- **Cyrillic Script (1 language)**: Bulgarian (bg);
- **Greek Script (1 language)**: Greek (el).

A.3 Tokenizer Compression Efficiency

In these experiments, we evaluate the compression of tokenizers using tokens per character (Tab. 9). We then compare our modular multilingual tokenizers—Sequential BPE (Tab. 10) and merged monolingual Unigram (Tab. 11)—showing compression comparable to both monolingual baselines and a standard multilingual tokenizer of similar size. Sequential BPE is trained with an alphabetical ordering of languages. We also report compression for frequency-based subtokenizers (Tab. 12).

[7] https://en.wikipedia.org/wiki/List_of_ISO_639_language_codes

A.3.1 Tokens per Character (↓)

Tab. 9 reports tokens per character across all 21 languages. Our tokenizers consistently achieve the lowest and most uniform values (0.21–0.27), while open-weight tokenizers are higher and more variable, especially on the low-resource languages.

A.3.2 BPE Tokenizer

Tab. 10 reports NSL scores for BPE tokenizers across all 21 languages. Open-weight tokenizers show consistently high NSL. For similar vocabulary budgets, sequential BPE matches standard multilingual BPE (Multi 21), showing that modular construction does not hurt compression, while enabling extraction of monolingual subtokenizers that match independently trained monolingual tokenizers at a fraction of the global vocabulary size.

A.3.3 Unigram Tokenizer

Tab. 11 reports NSL scores for Unigram tokenizers. Results are consistent with BPE: for similar vocabulary budgets, merged monolingual Unigram tokenizers achieve compression comparable to standard multilingual Unigram (Multi 21), while the extracted monolingual subtokenizers match independently trained monolingual tokenizers across all languages.

A.3.4 Frequency-based Subtokenizers

Tab. 12 reports NSL scores for frequency-based subtokenizers extracted from multilingual Unigram tokenizers of two sizes (256k and 384k). As k increases, compression approaches the full tokenizer, with $k = 24$k already achieving comparable compression. However, compression is inherently bounded by the base tokenizer — languages poorly compressed by it remain so regardless of k. This limitation motivates our focus on the merged monolingual approach, which matches independently trained monolingual tokenizers.

		NSL (↓)										
Model	Size	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU
TinyAya	261k	1.38	1.22	1.18	1.11	1.42	0.98	1.09	1.34	1.35	1.06	1.39
NLLB	256k	1.33	1.32	1.16	1.28	1.51	1.14	1.19	1.27	1.30	1.21	1.31
Gemma3	262k	1.42	1.38	1.29	1.13	1.73	0.98	1.07	1.48	1.54	1.09	1.49
Llama3	128k	1.78	1.36	1.45	1.35	1.78	0.98	1.27	1.75	1.83	1.24	1.87
Qwen3	152k	1.92	1.86	1.44	1.32	3.90	0.96	1.25	1.74	1.82	1.22	1.87
Multi 21	128k	1.16	1.07	1.07	1.10	1.17	1.06	1.08	1.11	1.14	1.09	1.12
	256k	1.04	0.96	0.99	1.00	1.04	0.98	1.00	1.00	1.02	1.01	1.01
	384k	0.99	0.91	0.95	0.95	0.98	0.96	0.97	0.94	0.95	0.97	0.96
	512k	0.96	0.88	0.93	0.93	0.94	0.94	0.95	0.91	0.91	0.95	0.92
Seq. 8k	116k	1.18	1.13	1.10	1.13	1.19	1.10	1.12	1.13	1.16	1.10	1.16
Seq. 16k	241k	1.05	1.01	1.00	1.02	1.05	1.01	1.02	1.02	1.03	1.01	1.04
Seq. 24k	366k	1.00	0.95	0.96	0.97	0.98	0.97	0.97	0.96	0.97	0.97	0.98
Extracted 24k	24k	1.00	1.00	1.01	1.02	0.99	1.01	1.01	1.02	1.05	1.01	1.04

Model	Size	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
TinyAya	261k	1.09	1.40	1.45	1.19	1.32	1.10	1.25	1.28	1.30	1.21
NLLB	256k	1.22	1.30	1.27	1.23	1.40	1.21	1.29	1.23	1.23	1.22
Gemma3	262k	1.16	1.59	1.72	1.24	1.33	1.12	1.30	1.43	1.42	1.29
Llama3	128k	1.38	2.03	2.10	1.43	1.67	1.31	1.53	1.62	1.62	1.46
Qwen3	152k	1.36	1.94	2.08	1.41	1.55	1.28	1.52	1.82	1.68	1.45
Multi 21	128k	1.09	1.11	1.11	1.09	1.11	1.08	1.08	1.06	1.09	1.07
	256k	1.00	1.00	1.01	1.00	1.00	0.99	1.00	0.96	0.99	0.99
	384k	0.97	0.94	0.95	0.96	0.94	0.96	0.95	0.91	0.94	0.94
	512k	0.95	0.90	0.92	0.94	0.90	0.94	0.93	0.88	0.91	0.92
Seq. 8k	116k	1.12	1.16	1.15	1.10	1.15	1.11	1.11	1.10	1.11	1.09
Seq. 16k	241k	1.02	1.03	1.03	1.01	1.03	1.00	1.01	0.99	1.01	0.99
Seq. 24k	366k	0.97	0.97	0.97	0.97	0.96	0.96	0.96	0.94	0.96	0.95
Extracted 24k	24k	1.02	1.04	1.03	1.03	1.03	1.01	1.02	1.04	1.04	1.03

Table 10: **BPE NSL** per language, relative to a 24k-token monolingual BPE tokenizer. *Multi 21*: standard BPE trained jointly on all 21 languages. *Seq. k*: the global sequential tokenizer with a budget of k tokens per language (Algorithm 2). *Extracted 24k*: per-language subtokenizers extracted from *Seq. 24k* (§ 4.2).

		NSL (↓)										
Model	Size	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU
TinyAya	261k	1.38	1.22	1.18	1.11	1.42	0.98	1.09	1.34	1.35	1.06	1.39
NLLB	256k	1.33	1.32	1.16	1.28	1.51	1.14	1.19	1.27	1.30	1.21	1.31
Gemma3	262k	1.42	1.38	1.29	1.13	1.73	0.98	1.07	1.48	1.54	1.09	1.49
Llama3	128k	1.78	1.36	1.45	1.35	1.78	0.98	1.27	1.75	1.83	1.24	1.87
Qwen3	152k	1.92	1.86	1.44	1.32	3.90	0.96	1.25	1.74	1.82	1.22	1.87
Multi 21	128k	1.18	1.04	1.09	1.07	1.15	1.05	1.06	1.20	1.13	1.07	1.09
	256k	1.10	0.97	1.02	1.00	1.07	1.00	1.00	1.09	1.02	1.01	1.01
	384k	1.08	0.96	1.01	0.98	1.06	1.00	1.00	1.06	1.00	1.00	1.00
	512k	1.08	0.96	1.01	0.98	1.06	1.00	1.00	1.06	1.00	1.00	1.00
Merged 8k	120k	1.18	1.12	1.09	1.12	1.20	1.10	1.10	1.12	1.13	1.10	1.13
Merged 16k	249k	1.06	1.00	1.00	1.01	1.06	1.01	1.01	1.01	1.01	1.01	1.01
Merged 24k	376k	1.00	0.94	0.96	0.97	0.99	0.98	0.97	0.96	0.95	0.98	0.97
Extracted 24k	24k	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Model	Size	NSL (↓) IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
TinyAya	261k	1.09	1.40	1.45	1.19	1.32	1.10	1.25	1.28	1.30	1.21
NLLB	256k	1.22	1.30	1.27	1.23	1.40	1.21	1.29	1.23	1.23	1.22
Gemma3	262k	1.16	1.59	1.72	1.24	1.33	1.12	1.30	1.43	1.42	1.29
Llama3	128k	1.38	2.03	2.10	1.43	1.67	1.31	1.53	1.62	1.62	1.46
Qwen3	152k	1.36	1.94	2.08	1.41	1.55	1.28	1.52	1.82	1.68	1.45
Multi 21	128k	1.07	1.17	1.26	1.08	1.06	1.06	1.09	1.08	1.14	1.08
	256k	1.00	1.07	1.14	1.01	0.98	1.00	1.02	1.01	1.06	1.00
	384k	1.00	1.06	1.12	1.00	0.97	0.99	1.01	1.00	1.04	0.99
	512k	0.99	1.06	1.11	1.00	0.96	0.99	1.01	1.00	1.04	0.99
Merged 8k	120k	1.09	1.14	1.14	1.10	1.15	1.10	1.11	1.11	1.11	1.10
Merged 16k	249k	1.00	1.01	1.02	1.01	1.02	1.00	1.01	1.00	1.00	1.00
Merged 24k	376k	0.97	0.95	0.96	0.97	0.96	0.97	0.96	0.94	0.95	0.96
Extracted 24k	24k	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 11: **Unigram NSL** per language, relative to a 24k-token monolingual Unigram tokenizer. *Multi 21*: standard Unigram trained jointly on all 21 languages. *Merged k*: concatenation of the 21 monolingual tokenizers of vocabulary size k, with scores re-estimated as in Algorithm 1. *Extracted 24k*: per-language subtokenizers extracted from *Merged 24k* (§ 4.1); all values round to 1.00 (true range 0.9999–1.0027).

Base Size	k	NSL (↓) BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU
256k	8k	1.19	1.26	1.19	1.21	1.21	1.17	1.18	1.21	1.23	1.17	1.20
	16k	1.10	1.08	1.07	1.07	1.09	1.05	1.06	1.12	1.09	1.06	1.07
	24k	1.10	1.02	1.04	1.03	1.08	1.02	1.03	1.10	1.05	1.03	1.03
	32k	1.10	1.00	1.03	1.01	1.07	1.01	1.01	1.09	1.03	1.02	1.02
	Full	1.10	0.97	1.02	1.00	1.07	1.00	1.00	1.09	1.02	1.01	1.01
384k	8k	1.19	1.28	1.19	1.22	1.22	1.18	1.19	1.22	1.24	1.17	1.21
	16k	1.10	1.08	1.08	1.07	1.09	1.06	1.06	1.12	1.09	1.06	1.07
	24k	1.09	1.02	1.04	1.03	1.07	1.02	1.03	1.09	1.05	1.03	1.03
	32k	1.08	1.00	1.03	1.01	1.06	1.01	1.01	1.08	1.03	1.02	1.02
	Full	1.08	0.96	1.01	0.98	1.06	1.00	1.00	1.06	1.00	1.00	1.00

Base Size	k	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
256k	8k	1.20	1.21	1.22	1.19	1.26	1.20	1.19	1.23	1.19	1.20
	16k	1.07	1.11	1.16	1.07	1.08	1.06	1.08	1.09	1.10	1.07
	24k	1.03	1.09	1.14	1.03	1.02	1.03	1.05	1.04	1.07	1.03
	32k	1.02	1.08	1.14	1.02	1.00	1.01	1.03	1.03	1.06	1.02
	Full	1.00	1.07	1.14	1.01	0.98	1.00	1.02	1.01	1.06	1.00
384k	8k	1.21	1.22	1.23	1.20	1.29	1.21	1.20	1.24	1.20	1.21
	16k	1.07	1.11	1.15	1.07	1.09	1.07	1.08	1.09	1.10	1.07
	24k	1.03	1.08	1.13	1.03	1.02	1.03	1.04	1.04	1.07	1.03
	32k	1.02	1.07	1.12	1.02	1.00	1.01	1.03	1.02	1.06	1.01
	Full	1.00	1.06	1.12	1.00	0.97	0.99	1.01	1.00	1.04	0.99

Table 12: **Frequency-based subtokenizers NSL score.** NSL per language, computed relative to a 24k tokens monolingual Unigram tokenizer. From multilingual Unigram tokenizers of various sizes, we extract the k most frequent tokens, as described in § 4.1, to build a language-specific tokenizer. We then compute the NSL scores for both the multilingual tokenizer and the language-specific one. We observe that when $k = 24$k, the language-specific subtokenizer achieves a compression performance similar to that of the original multilingual tokenizer (*Full*).

A.4 Sequential BPE Robustness

A.4.1 Robustness to Language Order in Training

To study the sensitivity of the resulting tokenizer to the order of languages in sequential BPE training, we train the tokenizer using multiple language orderings. Fig. 4 shows that the NLS of the resulting tokenizer varies only slightly across different training orders for all considered languages. This supports our choice of using alphabetical ordering for training in all experiments, as it provides a simple and consistent strategy without significantly impacting results.

A.4.2 Robustness against Naive Concatenation in Subtokenizer Construction

The experiment in Fig. 5 extends § 7. We compare two approaches for constructing subtokenizers for a subset of n languages: (i) combining, as described in § 4.2, the corresponding subtokenizers extracted from sequential BPE trained under different language orders, and (ii) a concatenation baseline that appends non-overlapping tokens and merges from independently trained monolingual tokenizers, where the resulting compression ratio varies significantly with the order of concatenation. Results in Fig. 5 show that sequential training yields more robust subtokenizer composition than naive concatenation.

A.5 Scaling of Vocabulary Size in Sequential Multilingual BPE with the Number of Languages

While ensuring fairness across languages, this sequential approach scales poorly as the language set expands. As illustrated in Fig. 6, the vocabulary size increases linearly with the number of languages. This trend underscores the need for training methods capable of integrating subtokenizers to maintain equitable compression ratios across a diverse linguistic range while staying computationally feasible. Our selection encompasses a highly diverse range of linguistic families, including major groups such as Indo-European, Afro-Asiatic, and Niger-Congo, as well as Dravidian, Turkic, Austronesian, Sino-Tibetan, and Tai-Kadai systems. This ensures our evaluation covers a broad spectrum of morphological structures.

A.6 Inference Time Comparison for Unigram Tokenizers

Fig. 7 reports normalized inference time for Unigram tokenizers. Extracted subtokenizers from the merged multilingual tokenizer achieve inference speeds comparable to independently trained monolingual tokenizers. Frequency-based subtokenizers also reduce inference time, but remain bounded by the compression of the base multilingual tokenizer, with diminishing returns beyond $v = 256$k.

B Experimental Settings

Model Architecture	
Number of non-embedding parameters	2.5B
Layers	24
Hidden dimension	3072
FFN latent dimension	8448
Number of heads	24
GQA	2
RoPE θ	100,000
Context length	4096
Activation function	SiGLU
Normalization	RMS pre-norm

Table 13: Model architecture details.

Hyperparameters	**Pre-training Settings**
Optimizer	
Optimizer Type	AdamW
Beta 1	0.9
Beta 2	0.95
ϵ	$1e{-8}$
Weight Decay	0.1
Gradient Clipping	1.0
Scheduler	
LR Scheduler	Cosine
Max LR	$3e{-4}$
Final LR	$3e{-5}$
Warmup Steps	2000
Number of Steps	300k
Batch Size	64
Training Tokens	78B

Table 14: Pre-training hyperparameters.

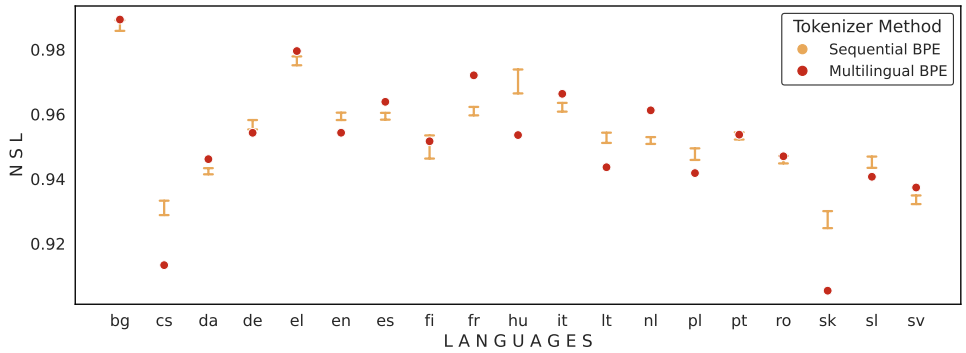
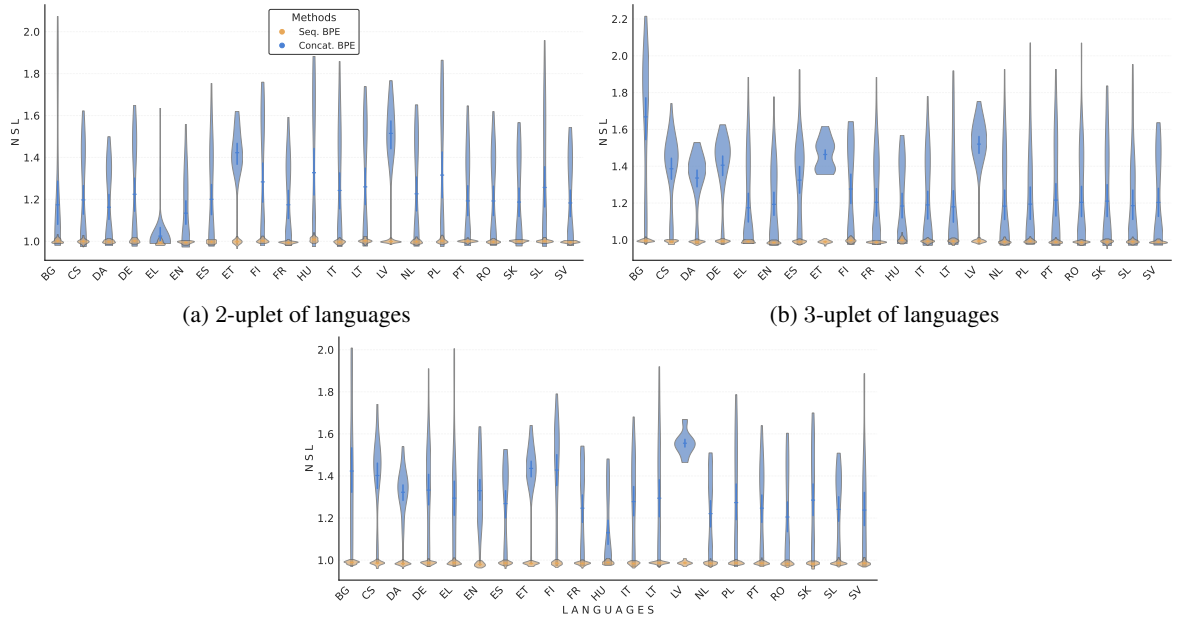


Figure 4: Per-language NLS comparison between a standard multilingual BPE tokenizer and a sequential one across language orders. In the sequential setting, each language is assigned a 24k token budget, yielding an average final vocabulary size of 361k tokens across orders, which matches the size of the standard BPE used for comparison. Horizontal bars show the min–max range across orders.



(a) 2-uplet of languages

(b) 3-uplet of languages

(c) 4-uplet of languages

Figure 5: Comparison per language of NLS for combinations of BPE tokenizers over p languages under different language orders, either via trivial concatenation (blue) or by combining the corresponding subtokenizers extracted from a sequential BPE (orange). The density (horizontal thickness) indicates the number of tokenizer combinations falling within each NLS range. It has been truncated where the density approaches zero for visual clarity.

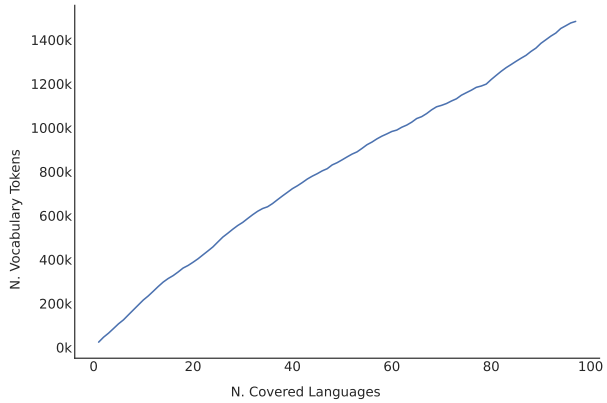


Figure 6: **Vocabulary size scaling** of a sequential multilingual BPE (24k tokens/language): despite shared tokens, the vocabulary grows near-linearly with the number of languages.

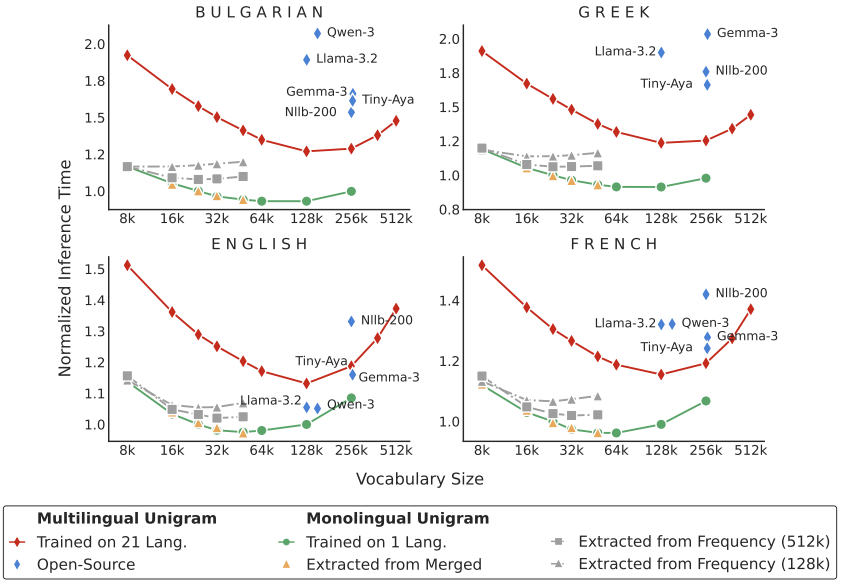


Figure 7: **Normalized inference time for Unigram Tokenizer**: We evaluate the inference time of a 2.5B parameter model (excluding embedding parameters) on FLORES-200 using both monolingual and multilingual Unigram tokenizers of varying vocabulary sizes. Monolingual tokenizers with a 24k vocabulary serve as the reference. We consider two strategies for deriving monolingual subtokenizers from multilingual tokenizers. First, *Extracted from Merged* corresponds to subtokenizers obtained from a multilingual Unigram built by merging independently trained monolingual tokenizers, followed by probability renormalization, as described in § 4.1. Second, *Extracted from Frequency* v constructs subtokenizers of size k by selecting the k most frequent tokens from a multilingual tokenizer of size v, as described in § 4.1. The curves for $v = 256$k and $v = 512$k overlap (not shown for clarity), showing diminishing returns when increasing the multilingual tokenizer size from $v = 256$k to $v = 512$k, and that subtokenizer compression remains bounded by the global tokenizer.

C Multilingual Performance

C.1 Subtokenizer robustness at inference time.

The $Samp_{\mathrm{Sub}\,n}$ training method is designed to support monolingual subtokenizers, enhancing both fairness and efficiency across diverse languages. This approach also accommodates subtokenizers covering specific language groups, which is particularly beneficial for tasks like translation. In Fig. 8, we highlight that training with $Samp_{\mathrm{Sub}\,2}$ (in Unigram setting) enables the use of various subtokenizers at inference time covering between 2 and 10 languages with minimal variations on downstream performance. Notably, the score variance remains low, and average performance closely tracks the baseline that utilizes the full tokenizer for both training and inference.

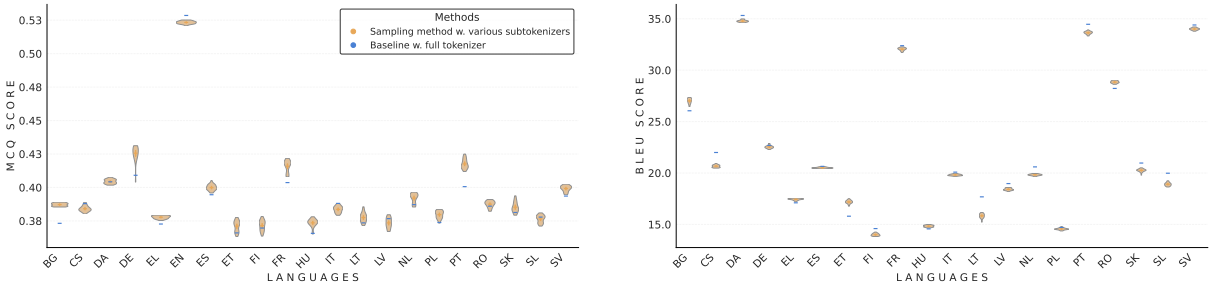
C.2 BPE Experiments

Tab. 15 presents MCQ and MT results for BPE tokenizers following the same setting as Tab. 6, confirming that our modular tokenizer approach generalizes across both tokenization schemes. Detailed per-language results are reported in Tab. 16.

C.3 Translation Performance

Tab. 17 evaluates translation across more diverse language pairs to assess language switching. We modify the training distribution to 40% English and 60% other languages to improve non-English translation, keeping all other hyperparameters unchanged. We compare $Samp_{\mathrm{Sub}\,5}$ with the baseline, both using a Unigram tokenizer.

Both models show improved performance when translating directly between non-English languages. However, results vary depending on the language pair: $Samp_{\mathrm{Sub}\,5}$ sometimes matches the baseline, but in other cases performs worse. Analysis of generated outputs shows that even the baseline struggles to correctly switch languages, which is consistent with prior work relying on English as a pivot for such translations (Imamura et al., 2023). These limitations are primarily due to insufficient training. Direct translation between non-English languages requires strong cross-lingual alignment and reliable language switching, which our models have not yet fully acquired. In practice, strong performance on this task typically depends on training with substantially more tokens.



(a) Multi-Choice Questioning

(b) Machine Translation

Figure 8: Comparison of the baseline and our sampling training at inference. Both are trained using Unigram tokenizers. We use the $Samp_{\text{Sub}\,2}$ method at training time but evaluated it using subtokenizers covering between 2 and 10 languages to underline the robustness at inference time of this method.

Model	**Infer.**	**EN**	**BG**		**DA**		**EL**		**FR**		**IT**		**SK**	
		MCQ	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT	MCQ	MT
Baseline	Full	**52.6**	37.2	28.0	39.4	**36.1**	36.9	**18.5**	39.2	**32.8**	38.7	**20.8**	38.7	**22.6**
Mono-only	Lang	52.4	**39.6**	0.8	40.1	2.5	**38.4**	1.5	39.6	9.6	39.2	4.8	37.7	0.9
	Full	50.1	37.4	0.7	38.8	2.4	36.7	1.0	37.1	6.3	37.4	2.5	35.2	0.9
$Samp_{\text{Full}}$	Lang	52.3	38.3	27.9	39.7	35.3	38.3	18.1	**40.1**	31.9	39.2	20.0	38.7	20.2
	Full	52.5	38.1	27.9	39.7	35.7	37.6	18.2	39.6	31.7	39.1	20.1	38.5	20.8
$Samp_{\text{Sub}\,3}$	Lang	52.5	38.6	**28.1**	39.9	36.0	38.0	18.3	39.8	32.5	**39.4**	20.3	**39.0**	21.4
	Full	52.4	38.2	27.7	**40.2**	**36.1**	37.8	18.3	39.4	32.3	38.8	20.6	38.5	21.7

Table 15: MCQ and MT performance under different pretraining sampling schemes for BPE tokenizers, following the same setting as Tab. 6. *Infer.* denotes the tokenizer used at inference; *Full* corresponds to the full tokenizer; *Lang* to the monolingual tokenizer for MCQ and the combined English–target tokenizer for MT.

Lang.	**Tasks**	Baseline	Lang.			Full		
			$Samp_{\text{Sub}\,3}$	$Samp_{\text{Sub}\,5}$	$Samp_{\text{Full}}$	$Samp_{\text{Sub}\,3}$	$Samp_{\text{Sub}\,5}$	$Samp_{\text{Full}}$
BG	ARC-E	51.3	**55.0**	54.5	54.0	53.1	54.9	54.0
	ARC-C	31.9	**32.6**	32.3	32.1	31.2	31.6	32.0
	HellaSwag	47.1	**48.3**	47.5	48.0	47.3	48.2	47.6
	MMLU	28.5	28.9	29.0	**29.7**	29.1	29.1	29.1
	BeleBele	27.1	28.3	27.7	**28.6**	27.9	27.8	27.7
	Flores	28.0	28.1	27.7	28.1	**28.1**	27.9	27.9
	Avg.	35.7	**36.9**	36.4	36.7	36.1	36.6	36.4
CS	ARC-E	55.1	55.4	54.9	**55.9**	54.3	54.5	54.9
	ARC-C	33.2	31.5	30.7	30.5	29.3	**34.0**	32.0
	HellaSwag	47.9	48.0	47.3	48.3	47.9	**48.7**	48.2
	MMLU	**29.8**	29.4	29.0	29.4	29.0	29.0	29.7
	BeleBele	29.0	**31.8**	30.3	30.4	29.7	30.9	28.3
	Flores	**22.3**	21.5	21.2	22.3	21.9	20.9	20.8
	Avg.	36.2	36.3	35.6	36.1	35.4	**36.3**	35.7
DA	ARC-E	56.2	56.9	**57.1**	55.3	56.2	56.0	56.6
	ARC-C	32.2	32.1	**33.3**	32.3	31.8	32.8	33.0
	HellaSwag	50.7	**51.5**	50.6	50.1	49.5	50.6	50.2
	MMLU	29.5	29.1	**29.8**	29.1	29.2	28.9	29.3
	BeleBele	28.6	30.0	**30.2**	30.0	30.0	**30.2**	29.6
	Flores	**36.1**	36.0	36.1	35.7	36.0	35.3	35.7
	Avg.	38.9	39.3	**39.5**	38.7	38.8	39.0	39.1

Lang.	Tasks	Baseline	Lang. $Samp_{\text{Sub 3}}$	Lang. $Samp_{\text{Sub 5}}$	Lang. $Samp_{\text{Full}}$	Full $Samp_{\text{Sub 3}}$	Full $Samp_{\text{Sub 5}}$	Full $Samp_{\text{Full}}$
	ARC-E	53.3	**53.7**	53.3	52.7	52.1	53.0	52.4
	ARC-C	30.9	30.9	31.7	31.8	31.2	**34.8**	34.0
	HellaSwag	45.8	**45.9**	45.0	45.3	45.0	45.8	45.7
DE	MMLU	29.3	29.5	**30.3**	28.9	29.2	29.0	29.1
	BeleBele	28.6	28.2	28.0	**29.7**	29.2	28.6	27.8
	Flores	22.6	22.5	22.3	23.0	**23.2**	21.6	21.7
	Avg.	35.1	35.1	35.1	35.2	35.0	**35.4**	35.1
	ARC-E	49.7	51.3	50.7	50.9	50.2	52.6	**52.7**
	ARC-C	31.7	32.1	33.1	**33.8**	32.3	33.5	31.1
	HellaSwag	46.0	**47.1**	46.6	46.8	46.5	46.5	46.2
EL	MMLU	28.1	29.3	**29.5**	29.1	28.6	28.8	29.3
	BeleBele	28.9	30.1	28.9	27.9	28.1	**30.2**	28.9
	Flores	18.5	18.3	18.3	**19.0**	19.0	18.1	18.2
	Avg.	33.8	34.7	34.5	34.6	34.1	**35.0**	34.4
	ARC-E	66.6	67.7	**68.0**	67.3	67.4	66.3	67.3
	ARC-C	39.2	38.3	38.3	40.3	39.3	40.2	**40.9**
	CSQA	**61.5**	60.1	60.0	60.6	60.5	60.4	59.1
	HellaSwag	62.6	**62.8**	62.3	62.1	61.7	62.1	62.1
EN	PIQA	75.1	75.8	75.5	**76.4**	76.0	74.7	75.4
	PIQA	49.0	50.1	49.4	49.4	49.6	50.5	**50.7**
	MMLU	34.1	**34.3**	34.2	33.3	33.1	33.7	33.4
	BeleBele	**32.8**	31.2	31.4	32.3	32.2	30.6	31.0
	Avg.	52.6	52.5	52.4	**52.7**	52.5	52.3	52.5
	ARC-E	55.7	56.6	56.1	55.9	55.6	**57.0**	56.7
	ARC-C	33.7	34.3	33.9	33.3	32.0	33.4	**34.6**
	HellaSwag	48.8	**49.3**	48.5	49.1	48.1	49.1	48.8
ES	MMLU	29.8	**30.4**	30.2	29.8	30.4	30.0	29.4
	BeleBele	29.0	**32.2**	29.7	30.0	30.3	30.7	30.1
	Flores	**21.6**	20.8	20.7	21.2	21.2	20.5	20.8
	Avg.	36.4	**37.3**	36.5	36.5	36.3	36.8	36.7
	ARC-E	49.6	50.1	48.4	49.6	48.8	**50.9**	50.7
	ARC-C	31.2	30.5	**31.5**	30.5	28.6	30.6	30.6
	HellaSwag	45.2	45.8	45.1	**46.0**	44.9	45.7	45.5
ET	MMLU	28.8	**29.0**	28.9	28.5	28.4	28.9	28.4
	BeleBele	27.6	27.7	28.3	27.2	27.7	28.4	**29.8**
	Flores	17.2	17.4	**17.4**	16.9	17.3	16.5	16.4
	Avg.	33.2	33.4	33.3	33.1	32.6	33.5	**33.6**
	ARC-E	51.3	**51.5**	49.2	50.8	49.3	51.3	**51.5**
	ARC-C	29.4	30.4	**30.8**	29.5	29.8	30.4	30.2
	HellaSwag	46.1	46.9	45.3	**46.9**	45.8	46.5	46.1
FI	MMLU	28.5	**29.1**	28.3	28.6	28.9	28.7	28.8
	BeleBele	28.3	28.9	27.9	**29.3**	28.2	29.0	29.0
	Flores	15.5	15.3	**15.6**	15.5	15.4	15.0	14.4
	Avg.	33.2	**33.7**	32.8	33.4	32.9	33.5	33.3
	ARC-E	54.8	55.6	55.5	54.5	54.0	**56.1**	55.6
	ARC-C	32.2	33.0	32.5	33.6	32.5	**34.2**	32.5
	HellaSwag	**49.6**	49.5	48.6	49.4	48.9	49.2	48.9
FR	MMLU	**30.0**	29.8	29.7	29.2	29.5	30.0	29.5
	BeleBele	29.6	31.3	30.4	31.6	30.6	31.1	**31.7**
	Flores	32.8	32.5	32.3	**33.0**	33.0	31.9	31.7
	Avg.	38.1	38.6	38.2	38.6	38.1	**38.7**	38.3
	ARC-E	51.0	49.9	49.5	**51.3**	50.3	50.5	50.6
	ARC-C	30.4	29.3	30.3	**32.4**	30.3	31.8	30.6
	HellaSwag	44.0	44.3	43.6	44.4	43.5	**44.5**	44.1
HU	MMLU	29.1	29.1	28.5	28.5	**29.2**	29.2	28.7
	BeleBele	28.7	**30.0**	29.7	**30.0**	28.9	28.8	29.2
	Flores	15.0	15.1	14.8	15.0	**15.4**	14.2	14.5
	Avg.	33.0	33.0	32.7	**33.6**	32.9	33.2	33.0
	ARC-E	52.9	54.8	54.0	54.4	52.9	55.8	**56.0**
IT	ARC-C	**35.0**	32.6	32.8	32.7	33.4	32.8	32.2
	HellaSwag	47.8	**48.7**	47.7	48.2	47.7	48.3	48.2
	MMLU	30.0	**30.5**	29.7	28.9	28.8	29.9	29.3
	BeleBele	27.9	**30.4**	29.8	30.0	**30.4**	29.3	29.8
	Flores	**20.8**	20.3	20.6	20.5	20.7	20.0	20.1
	Avg.	35.7	**36.2**	35.8	35.8	35.7	36.0	35.9

Lang.	Tasks	Baseline	Lang.			Full		
			$Samp_{\text{Sub 3}}$	$Samp_{\text{Sub 5}}$	$Samp_{\text{Full}}$	$Samp_{\text{Sub 3}}$	$Samp_{\text{Sub 5}}$	$Samp_{\text{Full}}$
LT	ARC-E	52.9	53.3	52.6	53.5	53.5	**54.5**	54.2
	ARC-C	31.9	29.9	31.7	30.0	30.4	**32.8**	31.7
	HellaSwag	46.5	46.4	45.6	**46.8**	45.5	46.5	46.0
	MMLU	29.3	29.2	28.9	**29.3**	28.7	28.8	28.9
	BeleBele	27.7	28.6	28.3	28.4	27.2	**29.0**	28.1
	Flores	17.9	**18.4**	18.3	17.6	17.7	17.7	18.1
	Avg.	34.4	34.3	34.2	34.3	33.8	**34.9**	34.5
LV	ARC-E	53.2	53.4	51.4	53.5	51.3	**53.5**	53.1
	ARC-C	31.1	33.3	31.4	31.7	30.0	**33.4**	32.8
	HellaSwag	46.3	**46.9**	46.0	46.5	45.6	46.5	46.5
	MMLU	28.8	29.1	28.5	29.1	27.9	**29.7**	28.1
	BeleBele	27.9	29.9	**30.0**	28.3	28.1	27.2	29.0
	Flores	20.2	19.9	19.8	20.4	**20.4**	19.6	19.9
	Avg.	34.6	**35.4**	34.5	34.9	33.9	35.0	34.9
NL	ARC-E	53.2	53.4	52.5	52.3	51.4	**54.6**	53.7
	ARC-C	29.3	31.0	**32.8**	31.1	31.1	32.3	30.5
	HellaSwag	47.7	47.7	47.1	48.2	47.6	**48.5**	48.3
	MMLU	29.0	28.9	**29.5**	28.5	29.3	28.5	28.9
	BeleBele	30.0	31.3	30.4	**31.7**	30.8	31.1	30.3
	Flores	**20.9**	20.5	20.3	20.3	20.2	20.2	20.1
	Avg.	35.0	35.5	35.4	35.3	35.1	**35.9**	35.3
PL	ARC-E	**53.9**	51.6	50.8	51.1	49.8	52.9	53.0
	ARC-C	32.5	30.8	**33.5**	32.3	32.3	31.7	33.2
	HellaSwag	46.6	46.5	45.9	**47.0**	46.1	46.8	46.6
	MMLU	28.6	28.6	28.3	28.6	28.1	**29.0**	28.3
	BeleBele	30.3	30.7	27.4	30.1	29.7	30.6	**31.2**
	Flores	**15.2**	14.5	14.4	14.4	14.7	14.0	14.3
	Avg.	**34.5**	33.8	33.4	33.9	33.4	34.2	34.4
PT	ARC-E	53.9	**54.3**	54.1	53.9	53.4	53.6	53.5
	ARC-C	33.8	31.8	33.0	**34.2**	33.0	32.6	31.9
	HellaSwag	47.6	**48.3**	47.0	48.2	46.9	48.0	48.0
	MMLU	29.6	29.8	29.7	**29.9**	29.8	29.8	29.8
	BeleBele	27.9	29.3	29.0	**30.0**	29.7	29.0	29.2
	Flores	**35.5**	34.3	34.3	34.6	34.9	33.5	34.2
	Avg.	38.0	38.0	37.9	**38.5**	37.9	37.8	37.8
RO	ARC-E	54.2	54.5	53.3	54.0	**54.8**	53.4	54.4
	ARC-C	30.5	30.7	31.5	**32.1**	31.7	31.2	30.2
	HellaSwag	47.6	**48.2**	47.0	47.8	46.6	48.1	47.8
	MMLU	29.7	**30.2**	30.2	29.4	28.9	29.9	29.8
	BeleBele	28.4	**29.3**	28.1	29.2	28.2	28.3	28.0
	Flores	**29.8**	28.6	28.7	28.9	29.0	28.3	28.8
	Avg.	36.7	**36.9**	36.4	36.9	36.5	36.5	36.5
SK	ARC-E	53.4	**54.8**	53.7	53.8	53.4	53.9	53.8
	ARC-C	**32.7**	31.9	**32.7**	32.3	31.4	31.3	30.3
	HellaSwag	47.5	47.6	46.9	47.5	47.0	**47.7**	47.6
	MMLU	29.3	29.0	28.6	**30.1**	29.4	29.5	29.5
	BeleBele	30.9	**31.6**	30.6	31.1	29.9	31.0	31.4
	Flores	**22.6**	21.4	21.7	21.7	21.9	20.2	20.8
	Avg.	36.1	36.0	35.7	**36.1**	35.5	35.6	35.6
SL	ARC-E	53.5	**54.1**	52.7	52.9	52.6	53.7	53.5
	ARC-C	32.3	31.2	31.8	30.6	30.5	**32.9**	31.7
	HellaSwag	46.7	**47.1**	46.0	46.2	45.7	46.8	46.3
	MMLU	28.5	29.2	28.8	28.4	28.3	28.9	**29.5**
	BeleBele	29.9	29.9	27.6	29.9	**30.1**	29.1	29.4
	Flores	19.7	19.7	19.8	19.6	**19.9**	18.8	19.2
	Avg.	35.1	**35.2**	34.4	34.6	34.5	35.0	34.9
SV	ARC-E	54.9	**56.4**	54.7	53.7	53.5	54.5	54.9
	ARC-C	**34.6**	32.0	33.4	32.7	30.8	32.2	31.8
	HellaSwag	49.4	**49.7**	49.0	48.9	48.1	49.2	49.7
	MMLU	29.3	**30.0**	29.9	28.8	29.2	29.6	29.4
	BeleBele	30.2	29.8	29.3	29.1	29.0	**30.4**	30.3
	Flores	34.2	34.2	34.0	34.5	**34.6**	33.6	33.9
	Avg.	**38.8**	38.7	38.4	38.0	37.5	38.2	38.3

Table 16: Detailed results for BPE experiments

→ Lang.	Model	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
BG	Baseline	-	**21.5**	**21.8**	**21.2**	**15.8**	**29.9**	**17.5**	**17.3**	**16.3**	**21.6**	**13.7**	**16.6**	**17.0**	**18.4**	**16.1**	**18.0**	**22.6**	**21.5**	**22.2**	**18.8**	**21.1**
	$Samp_{\text{Sub 5}}$	-	19.3	1.8	19.2	0.5	29.6	16.6	14.2	13.8	15.9	5.2	11.8	16.9	18.0	10.8	13.9	19.5	8.3	20.9	1.2	8.4
CS	Baseline	**16.0**	-	**18.9**	**19.1**	**14.7**	23.4	**13.2**	**14.4**	10.3	**17.0**	**13.1**	**13.9**	**14.6**	**15.0**	**13.4**	**16.0**	**16.5**	**17.5**	**26.5**	9.3	**17.4**
	$Samp_{\text{Sub 5}}$	4.2	-	7.5	18.3	12.8	**23.6**	11.8	11.7	**11.0**	12.6	9.0	11.0	12.6	13.5	12.1	13.6	14.7	15.2	23.7	**14.6**	11.0
DA	Baseline	21.6	23.3	-	**25.2**	19.5	**38.1**	18.5	**19.6**	18.5	**24.1**	16.2	**19.5**	17.7	19.1	18.5	17.6	25.2	22.9	22.6	**20.4**	**33.5**
	$Samp_{\text{Sub 5}}$	**23.2**	**23.4**	-	25.1	**20.2**	37.4	**18.8**	18.3	**19.2**	22.8	**17.9**	18.6	**18.1**	**19.2**	**19.2**	**17.6**	**26.4**	**24.4**	**23.1**	19.9	32.5
DE	Baseline	**17.3**	**19.8**	**21.2**	-	**15.3**	**25.1**	**14.5**	**15.8**	**13.9**	**18.9**	13.1	**15.8**	**14.1**	**15.1**	**17.0**	**14.9**	**19.7**	**18.8**	**19.1**	**16.3**	**20.4**
	$Samp_{\text{Sub 5}}$	16.0	16.8	6.6	-	14.5	24.6	13.3	12.1	12.7	16.5	**13.9**	14.5	13.0	13.7	13.8	13.6	16.7	17.8	17.2	14.8	18.9
EL	Baseline	**11.0**	**13.4**	**14.9**	**14.4**	-	19.3	**13.3**	**10.0**	**9.4**	**16.0**	**7.2**	**13.1**	**8.5**	**10.5**	**10.3**	**9.7**	**15.4**	**13.0**	**12.9**	**11.6**	**14.1**
	$Samp_{\text{Sub 5}}$	5.9	11.2	13.9	12.1	-	**19.6**	10.8	7.8	7.6	11.9	5.8	7.0	7.7	9.4	9.2	8.1	13.6	11.5	11.3	8.3	11.4
EN	Baseline	**35.4**	**34.1**	**44.7**	**37.5**	**30.1**	-	**27.5**	**29.1**	**27.8**	**39.0**	**26.7**	**29.8**	**27.2**	**30.0**	**28.8**	**25.3**	**42.3**	**37.6**	**33.5**	**29.7**	**44.0**
	$Samp_{\text{Sub 5}}$	34.7	32.4	43.2	34.5	29.8	-	27.1	26.9	25.8	36.4	25.4	29.1	25.4	28.3	28.4	23.8	41.9	35.7	31.7	27.0	42.5
ES	Baseline	**16.2**	**16.9**	**17.4**	**17.6**	**16.1**	**22.2**	-	**14.0**	**14.1**	**21.0**	13.2	**19.1**	12.4	**13.1**	**14.1**	15.5	**19.7**	19.7	**16.9**	**14.9**	**17.1**
	$Samp_{\text{Sub 5}}$	16.0	15.5	9.3	16.8	15.2	21.7	-	11.1	13.2	20.3	**14.4**	18.0	**13.5**	12.7	13.4	**15.5**	17.8	**19.9**	16.2	13.2	15.2
ET	Baseline	**9.6**	12.1	**12.6**	**12.5**	5.5	**18.9**	**8.2**	-	10.7	**11.2**	3.5	6.4	6.1	8.6	**8.2**	8.5	**9.2**	8.5	9.2	**9.4**	**12.6**
	$Samp_{\text{Sub 5}}$	6.3	**12.9**	5.6	11.3	**7.6**	17.7	7.4	-	**12.2**	4.8	**7.5**	**7.7**	**12.7**	**9.4**	8.1	**10.6**	5.2	**9.3**	**10.3**	9.1	10.9
FI	Baseline	**9.0**	**12.2**	**13.7**	**13.2**	**7.9**	**17.1**	**9.6**	**13.2**	-	**12.2**	**7.2**	**9.9**	**10.3**	**11.8**	**9.6**	**9.1**	**12.1**	**11.2**	**12.3**	**10.6**	**13.6**
	$Samp_{\text{Sub 5}}$	3.7	11.1	10.8	11.8	6.4	16.7	8.9	4.5	-	7.8	5.2	7.3	8.6	10.8	7.5	8.6	7.8	9.5	10.5	5.7	6.8
FR	Baseline	**20.7**	20.9	**21.0**	20.1	**21.3**	**35.1**	18.5	**15.4**	15.2	-	12.6	18.6	13.3	14.4	**18.0**	15.7	20.6	27.0	**22.1**	**19.1**	**24.0**
	$Samp_{\text{Sub 5}}$	20.6	**20.9**	15.6	**22.2**	19.2	34.5	**20.2**	11.8	**17.1**	-	**17.0**	**23.5**	**17.5**	**14.9**	16.8	**19.2**	**29.7**	**28.9**	20.4	16.4	20.7
HU	Baseline	**10.6**	**13.3**	**13.7**	**14.9**	**8.7**	**16.6**	**11.1**	**11.1**	**8.6**	**12.7**	-	**10.9**	**8.2**	**9.9**	**10.8**	**8.8**	**13.3**	**12.0**	**12.0**	**12.2**	**13.6**
	$Samp_{\text{Sub 5}}$	2.3	9.6	10.6	12.5	0.7	16.1	8.5	4.2	6.8	7.4	-	6.0	8.1	7.7	8.2	8.3	6.5	7.1	9.2	9.7	7.6
IT	Baseline	**14.8**	**17.3**	**17.7**	**17.8**	**15.0**	**22.5**	**18.7**	**14.0**	**12.1**	**23.0**	**12.8**	-	**13.0**	**13.6**	**13.9**	**14.6**	**20.8**	**20.7**	**17.1**	**14.4**	**15.7**
	$Samp_{\text{Sub 5}}$	8.8	15.0	10.8	14.6	5.2	20.5	15.8	7.1	11.2	19.2	10.9	-	11.2	10.5	11.9	11.9	19.2	18.8	14.6	9.8	11.9
LT	Baseline	**8.3**	**12.4**	**13.4**	**14.6**	**7.7**	**19.6**	**11.2**	**12.1**	**10.8**	**14.4**	**2.7**	**7.6**	-	**14.7**	**9.1**	**10.4**	**8.7**	**10.2**	**9.4**	**7.1**	**12.7**
	$Samp_{\text{Sub 5}}$	0.7	12.2	11.9	13.2	0.4	19.2	8.5	1.3	9.3	6.7	1.1	1.9	-	8.1	7.7	6.1	4.3	9.3	6.2	2.2	9.9
LV	Baseline	**12.3**	10.9	**14.6**	**15.2**	**8.7**	**21.3**	**11.3**	**13.2**	**12.1**	**13.9**	4.9	**11.4**	**13.8**	-	9.4	9.8	**12.3**	11.3	11.4	**11.2**	**11.8**
	$Samp_{\text{Sub 5}}$	2.6	**14.0**	7.1	14.0	3.6	20.0	9.0	2.3	10.3	6.1	**8.2**	7.5	10.5	-	**9.7**	**10.0**	11.4	**11.7**	**13.0**	7.6	11.5
NL	Baseline	**14.7**	**16.2**	**18.5**	19.0	**13.7**	**21.6**	**13.6**	**13.1**	**13.3**	**16.6**	**12.3**	**14.5**	**12.3**	**13.0**	-	**13.7**	**16.4**	**16.0**	**16.2**	**14.1**	**17.9**
	$Samp_{\text{Sub 5}}$	11.2	14.7	7.8	**19.1**	11.3	21.3	12.6	10.6	11.1	13.5	11.0	13.6	11.7	12.0	-	12.9	15.5	14.6	14.8	9.2	14.8
PL	Baseline	**11.6**	**14.0**	**12.7**	**14.0**	**8.5**	**15.9**	**11.6**	**9.7**	**8.9**	**14.6**	**8.2**	**11.8**	**9.8**	**10.7**	**10.5**	-	**12.4**	**12.3**	**13.7**	**9.5**	**11.9**
	$Samp_{\text{Sub 5}}$	3.2	6.0	5.3	11.8	3.1	15.4	6.9	2.0	4.3	6.0	3.3	1.5	8.8	3.5	6.7	-	4.2	6.1	5.8	4.1	2.6
PT	Baseline	**23.6**	**22.9**	**26.4**	**24.8**	**21.2**	**37.6**	**22.9**	**20.1**	**18.4**	**31.2**	**17.7**	**24.8**	**18.1**	**19.2**	**18.9**	**18.3**	-	**28.1**	**23.0**	**20.1**	**25.5**
	$Samp_{\text{Sub 5}}$	16.9	20.5	18.2	23.7	13.4	36.7	21.3	15.3	17.1	28.6	15.3	23.0	15.5	15.7	16.9	16.4	-	27.6	19.8	17.3	21.2
RO	Baseline	**20.2**	**21.7**	**24.2**	**23.7**	**18.4**	**31.8**	**21.2**	**15.4**	**15.4**	**27.9**	**14.2**	**22.1**	**14.7**	**16.0**	**16.7**	**16.6**	**27.6**	-	**21.2**	**19.0**	**22.7**
	$Samp_{\text{Sub 5}}$	7.8	19.1	22.1	22.8	3.9	30.5	17.4	12.1	11.4	18.7	6.7	5.9	13.5	11.7	11.0	12.5	20.8	-	13.1	7.1	19.2
SK	Baseline	**17.6**	**27.1**	**19.4**	**20.3**	**14.0**	**24.4**	**14.2**	**14.2**	**12.6**	**18.6**	**12.1**	**15.3**	**14.2**	**15.1**	**14.2**	**15.3**	**18.2**	**16.9**	-	**16.9**	**19.3**
	$Samp_{\text{Sub 5}}$	4.5	23.3	16.0	18.6	13.0	22.9	11.9	9.5	10.0	14.5	7.7	10.9	13.5	11.9	11.2	12.6	14.9	13.4	-	10.3	14.6
SL	Baseline	**15.8**	**18.7**	**17.1**	**16.5**	**13.6**	**21.1**	**12.6**	**14.7**	**12.7**	**15.8**	**13.0**	**12.5**	**13.0**	**15.1**	**12.9**	**14.2**	**14.6**	**15.6**	**18.2**	-	**16.5**
	$Samp_{\text{Sub 5}}$	9.3	17.3	7.6	15.0	12.1	18.7	10.2	10.5	11.6	12.0	11.4	11.2	12.6	12.2	11.0	13.1	9.5	14.3	17.3	-	13.0
SV	Baseline	17.5	**22.0**	**32.2**	**24.2**	14.5	**37.2**	**16.5**	16.0	**18.5**	**22.8**	13.8	**17.3**	15.4	**16.5**	16.6	**14.8**	22.8	21.2	**20.9**	17.4	-
	$Samp_{\text{Sub 5}}$	**18.8**	20.7	7.4	23.9	**16.9**	36.2	16.5	**16.3**	16.8	20.8	**14.5**	16.5	**15.5**	16.3	**18.0**	14.7	**22.9**	**21.6**	19.4	**17.6**	-

Table 17: Machine translation results for all language-to-language directions. We compare the baseline against the sampling method. In both cases, we use the full tokenizer at inference.

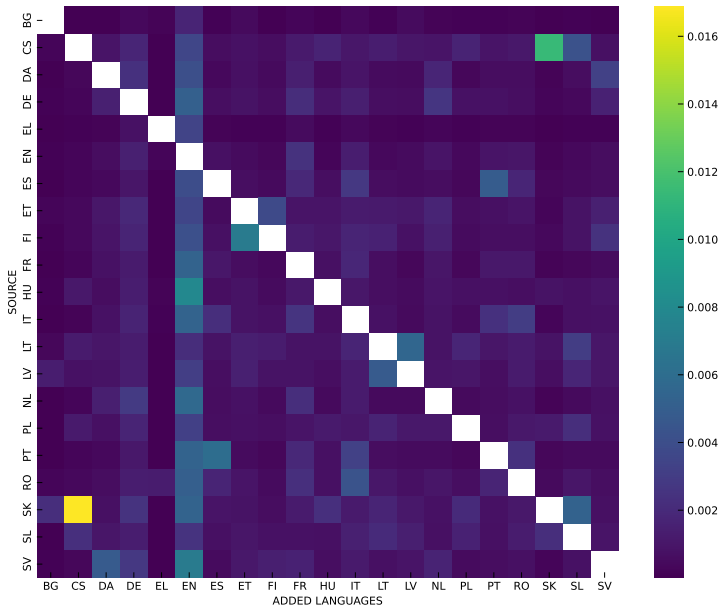


Figure 9: For each *Source* language, we compute the proportion of tokens that are exclusively assigned to each *Added Language* when tokenizing a text from the *Source* language with the *full tokenizer*.

D Supplementary Ablations

D.1 Language Origin of Tokens Produced by the Full Tokenizer

To determine the language origin of the tokens introduced during tokenization when processing text in a *Source* language with the *full tokenizer*, we select a subset of the training data for each source language. In Fig. 9, for each *Source* language, we compute the proportion of tokens that are exclusively assigned to each *Added Language* when tokenizing the text using the *full tokenizer*. We observe that English introduces the largest proportion of tokens across the considered languages, mainly due to its dominance in multilingual corpora.

D.2 Impact of the Sampling Probability

Tabs. 19 and 20 report the performance of $Samp_{\text{Full}}$ across different values of p, the probability of sampling the *full tokenizer* during training; the following conclusions also hold for $Samp_{\text{Sub}\,n}$.

As observed in § 6.2, $p = 0$ matches the baseline performance on MCQ tasks across all languages when inference is performed using the corresponding monolingual subtokenizer. However, this setting struggles to handle combinations of subtokens for MCQ and performs even worse on translation.

The case $p = 1$ corresponds to a setting similar to the baseline, where the model is trained exclusively using the *full tokenizer*. We observe baseline-level performance on translation when using both the *full tokenizer* and the combination (*EN + LANG*). The modularity of our tokenizer allows us to select the most relevant tokens for each language, leading to strong performance. However, a performance gap remains on MCQ tasks between the *full tokenizer* and the monolingual subtokenizers. This highlights the importance of sampling monolingual subtokenizers during training, in order to enable the model to handle arbitrary combinations of subtokenizers at inference time. In practice, training in this setting is computationally challenging due to the large vocabulary size, especially when scaling to many languages.

We also observe that lower values of p favor better performance when using monolingual subtokenizers, while higher values improve performance when using the *full tokenizer* (which serves as a proxy for inference with combined subtokenizers). We therefore selected $p = 0.5$ in our experiments to balance these two effects.

Additionally, lower values of p can help reduce training costs, as they decrease the proportion of batches in which logits must be computed over combined vocabularies, particularly for large values of n, and instead restrict computation to smaller, language-specific vocabularies. However, this effect is dependent on model size and is more pronounced in smaller-scale models, where the cost of computing logits represents a larger fraction of the overall computation.

Lang.	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
p (%)	0.5	3.5	1.9	2.2	0.7	1.3	2.2	2.3	2.8	1.7	2.2	2.4	2.8	2.5	2.0	2.2	2.5	2.6	4.8	2.3	2.7

Table 18: Percentage of tokens originating from other languages when tokenizing text in each reported language using the full tokenizer.

	Infer.	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
Baseline	Lang	37.3	38.8	40.4	37.9	37.3	52.9	39.5	36.6	37.0	39.5	36.6	38.8	37.4	37.7	38.7	37.4	38.3	38.6	38.1	37.8	39.4
	Full	37.3	38.8	40.4	37.9	37.3	52.9	39.5	36.6	37.0	39.5	36.6	38.8	37.4	37.7	38.7	37.4	38.3	38.6	38.1	37.8	39.4
0	Lang	38.3	**38.8**	40.5	37.7	37.9	52.6	39.9	37.3	**37.9**	40.3	37.0	38.9	37.4	**38.2**	39.4	37.9	38.4	38.5	**38.9**	37.7	39.7
	Full	36.3	36.6	38.3	36.3	37.1	49.6	38.0	34.4	34.2	37.8	35.3	37.2	34.6	35.1	37.3	35.1	37.7	36.2	36.1	36.0	38.2
0.1	Lang	**38.7**	38.4	41.3	38.2	**38.1**	52.4	**40.0**	37.0	37.7	39.3	**37.5**	38.8	37.5	37.6	39.4	37.7	38.4	**39.1**	38.3	37.8	39.6
	Full	38.2	38.0	40.6	37.5	37.5	52.1	39.0	35.8	36.5	38.9	36.6	38.1	35.9	36.1	38.6	36.6	38.0	37.8	37.8	36.9	38.9
0.2	Lang	38.0	37.6	39.7	37.4	37.7	52.2	39.4	36.5	36.9	39.2	36.5	38.2	37.3	37.4	38.7	37.5	38.2	37.7	38.0	37.5	39.3
	Full	37.5	37.9	39.1	37.3	36.6	51.8	38.1	35.8	35.9	38.8	35.9	37.5	36.7	36.4	37.8	36.8	37.2	37.6	37.8	36.9	38.8
0.5	Lang	38.1	38.4	**41.6**	38.4	38.0	53.2	39.8	**37.6**	37.6	39.3	36.9	**39.1**	38.1	38.2	**39.7**	**38.1**	**38.8**	38.5	38.3	38.2	40.1
	Full	38.0	38.2	41.5	38.1	37.9	53.0	39.7	36.7	37.6	39.4	36.8	38.8	**38.1**	37.6	39.3	37.8	38.2	38.2	38.2	37.8	39.7
0.8	Lang	38.5	38.1	40.3	37.9	37.5	52.9	39.5	36.7	37.2	39.4	36.7	38.7	37.4	37.6	38.9	36.9	38.4	38.2	38.5	**38.2**	**40.4**
	Full	38.6	38.6	40.8	38.0	36.9	**53.5**	39.7	36.2	37.3	**41.6**	36.3	38.7	37.7	37.9	39.6	37.5	38.4	38.5	38.7	38.0	40.0
1	Lang	37.4	35.5	38.8	36.6	37.3	50.3	37.0	34.5	34.7	37.6	35.2	36.2	34.8	35.6	37.1	35.2	35.9	36.2	34.9	35.3	37.7
	Full	37.4	38.8	40.8	**38.5**	37.9	52.5	39.8	36.5	37.3	39.5	36.6	38.3	37.1	37.9	38.9	37.1	38.5	38.7	38.1	37.7	39.9

Table 19: MCQ Performance of $Samp_{\text{Full}}$ for various probabilities of sampling the full tokenizer (Unigram).

	Infer.	BG	CS	DA	DE	EL	EN	ES	ET	FI	FR	HU	IT	LT	LV	NL	PL	PT	RO	SK	SL	SV
Baseline	Lang	26.1	22.0	35.3	22.7	17.1	-	20.6	15.8	14.6	32.4	14.6	20.1	17.7	19.0	20.6	14.7	34.5	28.2	21.0	20.0	34.4
	Full	26.1	22.0	35.3	22.7	17.1	-	20.6	15.8	14.6	32.4	14.6	20.1	17.7	19.0	20.6	14.7	34.5	28.2	21.0	20.0	34.4
0	Lang	1.0	1.8	2.8	2.6	5.3	-	6.2	1.5	1.1	15.0	1.5	6.9	1.0	1.0	2.6	1.4	5.2	9.3	1.6	2.0	2.6
	Full	0.9	1.5	2.4	2.1	2.5	-	2.8	1.1	0.8	7.4	0.9	3.0	1.0	0.9	2.3	1.0	4.5	4.8	1.4	2.2	2.3
0.1	Lang	23.3	19.3	33.3	19.9	16.4	-	18.3	13.5	12.8	29.5	12.1	18.6	13.6	15.6	18.9	13.0	29.4	26.4	18.5	17.7	31.9
	Full	23.7	19.4	33.1	20.1	16.4	-	18.6	14.7	13.2	29.3	11.9	18.7	15.7	17.5	18.8	13.2	30.9	26.0	18.8	18.0	31.8
0.2	Lang	26.0	19.9	33.9	21.6	16.1	-	19.6	15.4	14.0	30.8	13.9	18.6	16.0	17.1	19.2	12.2	31.1	26.3	19.8	18.2	33.2
	Full	25.7	20.3	34.3	21.5	16.1	-	19.7	16.3	13.9	30.5	14.2	18.7	16.6	18.7	19.3	13.5	31.8	26.5	20.0	18.5	33.2
0.5	Lang	**27.9**	20.2	35.1	21.1	16.9	-	20.6	15.9	14.9	32.5	13.9	19.8	16.9	18.6	19.9	14.0	33.3	28.3	20.3	19.2	33.8
	Full	27.7	20.5	35.2	21.1	16.8	-	20.8	16.2	15.0	32.6	13.9	20.2	17.8	19.5	20.1	14.3	33.7	28.4	21.1	18.9	34.2
0.8	Lang	27.6	20.6	34.1	22.5	17.2	-	20.3	16.8	14.3	31.5	14.7	19.3	17.2	18.3	19.5	14.2	33.1	27.5	19.6	19.3	32.6
	Full	27.4	21.2	34.8	22.8	17.1	-	20.7	17.0	15.1	31.7	14.7	19.8	17.5	19.2	19.7	14.3	33.7	28.4	20.7	19.6	33.0
1	Lang	27.5	20.6	36.0	22.7	17.3	-	20.6	17.1	14.6	32.8	15.3	20.1	17.3	19.0	20.0	13.8	33.8	28.1	20.0	18.8	34.4
	Full	27.3	**22.1**	**37.1**	**23.2**	**17.4**	-	**21.1**	**17.5**	**15.4**	**34.0**	**15.6**	**20.7**	**19.1**	**20.1**	**20.7**	**14.9**	**35.0**	**29.4**	**22.1**	**20.3**	**35.5**

Table 20: MT Performance of $Samp_{\text{Full}}$ for various probabilities of sampling the full tokenizer (Unigram).

E Offloading Vocabulary State to CPU: Proof of Concept

Setup. We benchmark our real training step — FSDP-sharded forward/backward with flash attention, activation rematerialization, and AdamW — on one node of 8×H100, at 1B, 2.5B, and a 7B configuration (d=4096, 32 layers). The vocabulary state resides in CPU RAM: an fp16 table holding the embedding rows of all 21 languages, with fp32 Adam moments and master weights host-side. A step activates the batch's input rows (∼24k, one language) plus the output rows of the K sampled subtokenizers; we measure K=1 (monolingual batches, half of training at p=0.5) and K=6 ($Samp_{\text{Sub}\,5}$). The host side is deliberately naive: unpinned memory, threaded-NumPy AdamW. We report batch 64 — the configuration our full-vocabulary training was memory-constrained to — and batch 128, which the freed vocabulary memory makes affordable.

Pipeline. Following ZeRO-Offload's delayed parameter update (Ren et al., 2021): while the GPU

Model (batch)	K	fwd/bwd	off-GPU	xfer	step w/ offload
1B (128)	1	1.37	0.32	0.05	1.43
1B (128)	6	1.53–1.55	1.15–1.24	0.15	1.72–1.74
2.5B (64)	1	1.47	0.54	0.07	1.55
2.5B (64)	6	1.59	1.63–1.88	0.23	2.03–2.38
2.5B (128)	1	2.90	0.31–0.51	0.06	2.96–3.00
2.5B (128)	6	3.13	2.20–2.65	0.21–0.26	3.76–3.89
7B (64)	1	3.62	0.65–0.83	0.10	3.73–3.76
7B (64)	6	3.77–3.80	1.98–2.19	0.30	4.09–4.21
7B (128)	1	7.11–7.14	0.65	0.10	7.23–7.24
7B (128)	6	7.39–7.46	3.71–4.72	0.32–0.39	7.82–7.85

Table 21: Offloading proof of concept (seconds/step, median of 12; ranges over repeat runs on different nodes). *fwd/bwd*: step time with all state GPU-resident; *off-GPU*: the vocabulary round trip alone; *xfer*: its two row transfers; *step w/ offload*: step time with the round trip pipelined in. K=1: monolingual batch; K=6: $Samp_{\mathrm{Sub}\,5}$.

computes step t, the CPU gathers the active rows, applies AdamW using the gradients of step $t-1$, and writes them back to the host table; the vocabulary rows thus lag one gradient step, a scheme whose convergence neutrality is established by Ren et al. (2021). The backbone updates synchronously on GPU. In our prototype the two row transfers (gradients to the CPU, updated parameter rows back) run between steps, since copies issued while a step is in flight are stream-ordered behind its kernels; production implementations overlap them with compute through pinned buffers and dedicated copy streams (Ren et al., 2021; Miao et al., 2021).

Results. Tab. 21 reports, per configuration, the step time, the full off-GPU round trip measured alone (gather, both transfers, CPU update, write-back), its transfer share, and the step time with the round trip pipelined in. The round trip fits within the step at every batch-128 configuration and every monolingual one; the sole overflow is $Samp_{\mathrm{Sub}\,5}$ at the memory-constrained 2.5B/batch-64 setting, where the naive CPU update alone (1.4–1.7 s) exceeds the 1.6 s step — the component a fused CPU optimizer accelerates several-fold. Transfers are 10–15% of the round trip throughout (0.05–0.39 s): the off-GPU cost is dominated by CPU compute, which the pipeline hides, not by data movement.

Caveats. These timings upper-bound the overhead: no pinned memory, no fused CPU optimizer, no transfer/compute overlap. Host-side times vary up to $\sim$2$\times$ across nodes without NUMA pinning (the upper ends of the off-GPU ranges). The delayed update applies to the vocabulary rows only.

F Inference and Memory Efficiency Across Scales: Details

Setup. Inference time is the mean forward pass over the full FLORES corpus, context 4096, batch 1, bf16, on H100 GPUs right-sized per scale: one GPU up to 12B, two (tensor-parallel) at 30B, where the monolingual reference runs on the same hardware, keeping ratios comparable. Model configurations are given in Tab. 23; the 2.5B configuration matches our trained models. Tokenizers: Gemma-3 (262k), Qwen-3 (152k), *Full* (our sequential multilingual BPE, 387k), and *Extracted* (its per-language 24k subtokenizers), all against dedicated 24k monolingual BPE references.We measure 16 of the 21 languages: the six low-resource ones and ten from the conservative set; the five omitted (ES, PT, IT, DA, NL) fall within the reported range, and their ratios at scale follow from their NSL (see below).

Inference time. *Full*'s overhead is nearly language-independent — within $\pm$0.04 of each scale's mean, reflecting its pure output-layer cost — and decays from $\sim$1.58$\times$ to $\sim$1.07$\times$ (Tab. 22); open-weight overheads contain an additional compression factor that persists at every scale and is largest on low-resource languages and non-Latin scripts. At 30B, where the output-layer factor has faded, the normalized time converges to the tokenizer's NSL (Gemma-3 on TE: 1.73 vs. an NSL of 1.61; on SI: 1.92 vs. 1.79) — Tab. 2 thus predicts large-scale inference cost for any language. *Extracted* matches the monolingual references (0.99–1.04) everywhere.

	Gemma-3 (262k)						Qwen-3 (152k)						Llama-3 (128k)					
Lang	0.5B	1B	2.5B	7B	12B	30B	0.5B	1B	2.5B	7B	12B	30B	0.5B	1B	2.5B	7B	12B	30B
BG	2.15	1.97	1.82	1.63	1.59	1.51	2.46	2.33	2.22	2.07	2.03	2.00	2.07	1.99	1.92	1.84	1.81	1.83
CS	2.10	1.92	1.76	1.58	1.56	1.46	2.40	2.27	2.15	2.02	2.00	1.91	1.60	1.53	1.47	1.41	1.42	1.39
DE	1.72	1.58	1.45	1.31	1.28	1.20	1.70	1.61	1.52	1.43	1.42	1.36	1.57	1.51	1.44	1.40	1.39	1.37
EL	2.63	2.41	2.22	1.98	1.97	1.85	5.03	4.74	4.51	4.19	4.15	4.03	2.08	1.99	1.91	1.82	1.84	1.83
EN	1.49	1.37	1.26	1.12	1.11	1.05	1.24	1.17	1.11	1.03	1.03	1.00	1.15	1.10	1.06	1.00	1.01	1.01
FR	1.66	1.52	1.40	1.25	1.22	1.17	1.57	1.48	1.41	1.31	1.31	1.26	1.45	1.38	1.33	1.28	1.27	1.27
HR	2.05	1.88	1.73	1.55	1.52	1.43	2.15	2.03	1.93	1.80	1.78	1.72	1.90	1.81	1.75	1.68	1.67	1.66
HU	2.25	2.06	1.90	1.69	1.67	1.59	2.40	2.27	2.16	2.01	2.00	1.92	2.18	2.08	2.00	1.93	1.92	1.90
NE	2.21	2.03	1.88	1.66	1.64	1.54	5.37	5.07	4.83	4.51	4.48	4.29	2.85	2.74	2.63	2.51	2.51	2.49
PL	2.03	1.86	1.71	1.53	1.52	1.41	2.01	1.90	1.80	1.68	1.66	1.61	1.95	1.88	1.80	1.73	1.73	1.69
RO	1.98	1.81	1.68	1.50	1.47	1.39	1.97	1.86	1.77	1.65	1.65	1.58	1.80	1.73	1.66	1.60	1.58	1.58
SI	2.72	2.50	2.29	2.04	2.03	1.92	6.95	6.56	6.21	5.82	5.76	5.58	7.82	7.47	7.15	6.88	6.89	6.85
SL	2.15	1.98	1.81	1.62	1.59	1.51	2.16	2.04	1.94	1.80	1.79	1.75	1.90	1.83	1.74	1.68	1.67	1.65
SO	2.36	2.17	1.99	1.78	1.75	1.65	2.26	2.13	2.03	1.89	1.87	1.82	2.05	1.97	1.89	1.82	1.80	1.80
SW	2.39	2.19	2.03	1.81	1.77	1.67	2.39	2.26	2.15	2.02	2.00	1.93	2.17	2.08	2.00	1.93	1.90	1.90
TE	2.46	2.26	2.09	1.84	1.82	1.73	8.18	7.72	7.34	6.86	6.70	6.56	8.69	8.31	7.97	7.65	7.57	7.64
	TinyAya (261k)						**Full (387k, ours)**						**Extracted 24k (ours)**					
BG	1.92	1.76	1.63	1.49	1.46	1.47	1.62	1.45	1.32	1.15	1.12	1.10	1.00	1.00	1.00	1.00	0.99	1.01
CS	1.71	1.56	1.44	1.32	1.31	1.29	1.55	1.39	1.25	1.11	1.09	1.05	1.00	1.00	1.00	1.00	1.01	1.00
DE	1.55	1.42	1.31	1.22	1.19	1.17	1.58	1.42	1.27	1.13	1.10	1.06	1.02	1.02	1.02	1.03	1.02	1.02
EL	1.99	1.82	1.67	1.54	1.51	1.50	1.60	1.43	1.29	1.13	1.11	1.08	1.00	1.00	1.00	0.99	1.00	1.00
EN	1.37	1.25	1.15	1.06	1.05	1.04	1.58	1.42	1.28	1.12	1.11	1.07	1.01	1.01	1.01	1.01	1.00	1.01
FR	1.48	1.36	1.25	1.14	1.14	1.13	1.58	1.42	1.28	1.12	1.10	1.07	1.01	1.00	1.01	1.00	1.00	1.01
HR	1.78	1.62	1.51	1.38	1.36	1.35	1.54	1.38	1.25	1.10	1.06	1.04	1.03	1.03	1.04	1.04	1.04	1.04
HU	1.92	1.76	1.62	1.49	1.46	1.45	1.59	1.43	1.29	1.13	1.11	1.08	1.03	1.03	1.03	1.03	1.03	1.03
NE	1.98	1.81	1.67	1.54	1.52	1.50	1.59	1.42	1.28	1.11	1.11	1.07	1.01	1.02	1.02	1.01	1.02	1.01
PL	1.84	1.68	1.55	1.43	1.42	1.38	1.57	1.42	1.27	1.12	1.10	1.06	1.03	1.03	1.03	1.03	1.04	1.03
RO	1.75	1.61	1.47	1.36	1.35	1.32	1.57	1.41	1.27	1.12	1.09	1.06	1.02	1.02	1.02	1.03	1.02	1.02
SI	12.7	11.6	10.6	9.78	9.71	9.65	1.55	1.39	1.25	1.09	1.09	1.05	1.00	1.00	1.00	1.00	1.00	1.00
SL	1.82	1.67	1.52	1.41	1.39	1.39	1.55	1.39	1.25	1.10	1.07	1.05	1.04	1.04	1.04	1.04	1.04	1.04
SO	2.26	2.06	1.90	1.75	1.72	1.72	1.57	1.41	1.27	1.11	1.09	1.06	1.02	1.02	1.02	1.02	1.02	1.02
SW	1.96	1.79	1.65	1.52	1.49	1.49	1.61	1.44	1.30	1.15	1.12	1.09	1.03	1.03	1.03	1.03	1.03	1.03
TE	2.35	2.15	1.99	1.82	1.78	1.78	1.58	1.41	1.27	1.12	1.09	1.07	1.00	1.00	1.00	1.00	0.99	1.01

Table 22: Inference-time grid: forward time on FLORES relative to a dedicated 24k monolingual tokenizer, per language and model scale (configurations in Tab. 23).

Memory. For parameters (Tab. 23): a 24k sub-vocabulary instead of a 256k one removes 90.6% of the embedding parameters at any width; the share of *total* parameters saved shrinks as the backbone grows, and is lower with tied embeddings (last two rows). For compression (Tab. 9): comparing an open-weight row with the *Extracted* row gives the sequence-length cut, which equally cuts decode steps and KV cache: on HR, Gemma-3 costs 0.314 tokens per character and ours 0.242, a 23% cut. Cuts span -3% (EN) to 44% (low-resource). These cuts hold at every size and architecture; only the per-token cache cost varies (58–745 KB from 0.5B to 30B; bf16, GQA 2, head dim. 128).

Scale	0.5B	1B	2.5B	7B	12B	30B
Width d	1536	2048	3072	4096	5120	7168
Layers	19	21	24	40	40	52
Total-parameter reduction, *256k → 24k vocab.:*						
untied emb.	55%	47%	35%	20%	16%	10%
tied emb.	40%	31%	21%	11%	9%	5%

Table 23: Benchmark configurations (scales named by non-embedding parameter count; head dim. 128, GQA 2, MLP $2.75d$; 2.5B matches our trained models) and the total-parameter reduction when serving a language with its 24k sub-vocabulary instead of a 256k one, for untied and tied embeddings.