Teaching Old Tokenizers New Words: Efficient Tokenizer Adaptation for Pre-trained Models

Taido Purason[1], **Pavel Chizhov**[2], **Ivan P. Yamshchikov**[2], **Mark Fishel**[1]
[1]Institute of Computer Science, University of Tartu
[2]CAIRO, Technical University of Applied Sciences Würzburg-Schweinfurt
taido.purason@ut.ee

Abstract

Tokenizer adaptation plays an important role in adapting pre-trained language models to new domains or languages. In this work, we address two complementary aspects of this process: vocabulary extension and pruning. The common approach to extension trains a new tokenizer on domain-specific text and appends the tokens that do not overlap with the existing vocabulary, which often results in many tokens that are unreachable or never used. We propose **continued BPE training** that extends a pre-trained tokenizer by continuing the BPE merge learning process on new data. Experiments across multiple languages and model families show that this approach improves tokenization efficiency and leads to better utilization of added vocabulary. We also introduce **leaf-based vocabulary pruning**, which removes redundant tokens while preserving model quality. Together, these methods provide practical tools for controlled vocabulary modification, which we release as an open-source toolkit.

taidopurason/tokenizer-extension

1 Introduction

When adapting large language models (LLMs) to new domains or languages, continued pre-training has become a widely used strategy. However, effective adaptation depends not only on updating model weights but also the tokenizer. Most LLMs rely on byte-pair encoding (BPE, Sennrich et al., 2016; Gage, 1994) tokenizers, and in practice their effectiveness varies substantially across languages. In particular, inefficient tokenization for underrepresented languages often leads to longer sequences and higher computational costs in LLMs (Petrov et al., 2023). One way to address this is to extend the vocabulary with domain-specific tokens, which improves compression and reduces sequence lengths, albeit at the cost of increasing the model size due to a larger vocabulary. Pruning infrequent or irrelevant tokens can address this trade-off by reducing vocabulary size. In this work, we focus on tokenizer modification of pre-trained models using those two steps: pruning and extension.

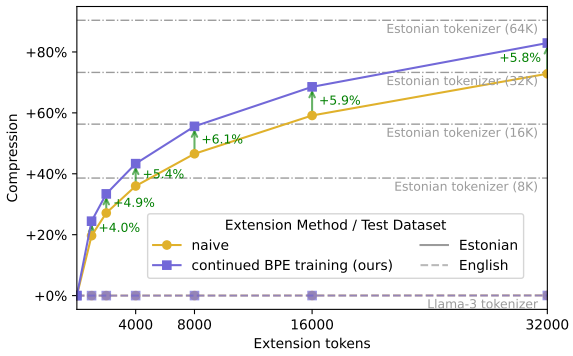

Figure 1: Change in text compression (*bytes per token* ↑) of text using the Llama 3 tokenizer extended with Estonian tokens using naive tokenizer extension and continued BPE training (ours). We also compare to a tokenizer trained from scratch on Estonian data.

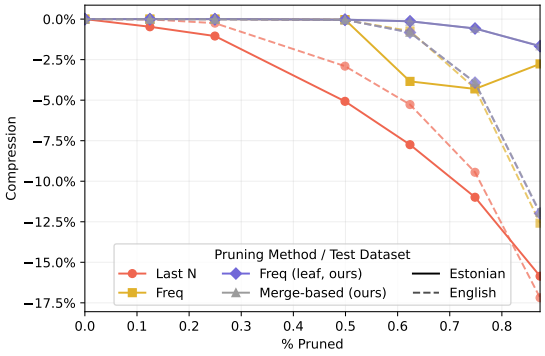

Figure 2: Change in text compression (*bytes per token* ↑) for vocabulary pruning methods on Estonian-English pruning (Llama-3). Note that *Freq (leaf)* and *Merge-based* method results overlap.

Extending an existing vocabulary is a widely used step in adapting pre-trained LLMs to new languages (Tejaswi et al., 2024; Csaki et al., 2024, 2023; Kiulian et al., 2025; Fujii et al., 2024; Lin et al., 2024; Yamaguchi et al., 2024; Cui et al.,

Findings of the Association for Computational Linguistics: EACL 2026, pages 6492–6516
March 24-29, 2026

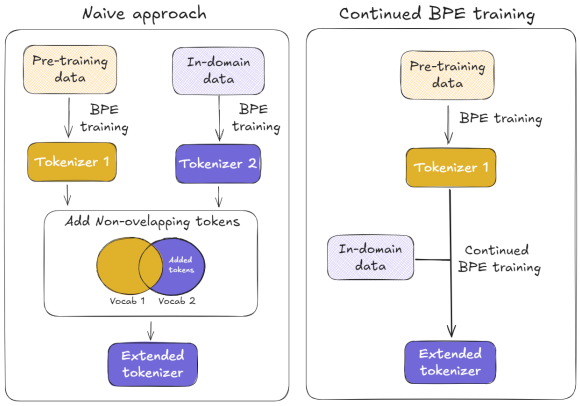

Figure 3: The comparison of vocabulary extension methods: previously common approach of training a tokenizer and using its vocabulary for extending an existing tokenizer (left) and continuing BPE training on an existing tokenizer proposed by us (right).

2024). In previous works, vocabulary extension typically involved training a new tokenizer on in-domain data and appending the tokens that were previously not present to the original tokenizer (see Figure 3). We show that this approach introduces useless tokens that do not participate in merges, resulting in lower compression compared to having tokens that are useful in tokenization. This happens because BPE strictly follows the merge sequence, e.g., when a word `information` is originally tokenized as `inform + ation`, adding new tokens `infor` and `mation` is useless, even though they are new to the vocabulary. In this work, we propose a more effective alternative: continuing to train the original BPE tokenizer on domain-specific data. We show that for the same number of added tokens, this approach yields better compression (see Figure 1). While we focus on LLMs, our method can be applied to other pre-trained models that use BPE tokenizers.

The other side of tokenizer adaptation, besides extension, is pruning. Thus, we also study the impacts of different tokenizer pruning strategies and introduce a non-invasive leaf-based pruning method that trims the vocabulary in a structure-preserving manner. This avoids creating unreachable tokens and yields higher tokenization efficiency (see Figure 2). By using token frequencies in a target corpus, akin to tokenizer training, we can preserve domain- and language-specific tokens without increasing sequence lengths in relevant domains while pruning irrelevant tokens. Our pruning method can also be used on its own in the context of model compression to reduce the vocabulary size, thereby substantially improving the model's efficiency: for instance, Gemma 3 (Kamath et al., 2025) has 270 million parameters, 170 million of which are used for vocabulary embeddings (Lacombe et al., 2025).

In summary, our contributions are threefold:

- First, we introduce **a new method for vocabulary extension** based on continued BPE training, and provide a detailed analysis showing that it yields more efficient tokenization than the widely used approach of adding tokens from an independently trained domain-specific tokenizer.
- Second, we develop **improved vocabulary pruning strategies** that do not change the tokenizer implementation and systematically analyze their effectiveness.
- Finally, **we release an open-source toolkit**[1] that supports modification of both Byte-level and SentencePiece BPE tokenizers for pre-trained models through Hugging Face `transformers` (HF, Wolf et al., 2020).

2 Related work

A growing body of work shows that tokenization quality is tied to model performance, highlighting its importance. Language-specific tokenizers have been shown to provide advantages over multilingual ones (Rust et al., 2021), and more efficient compression is associated with better downstream results (Goldman et al., 2024). Recent evidence also suggests that differences in tokenization quality may contribute to the performance gap observed between agglutinative and fusional languages (Arnett and Bergen, 2025).

2.1 Tokenizer extension

Several works adopted the strategy of training a new tokenizer on domain- or language-specific data and adding the resulting non-overlapping tokens to the original vocabulary (Tejaswi et al., 2024; Csaki et al., 2024, 2023; Kiulian et al., 2025; Fujii et al., 2024; Lin et al., 2024; Yamaguchi et al., 2024; Cui et al., 2024), which we refer to as the **naive approach**. MEDVOC-LLM (Balde et al., 2025) provides another way to add new tokens, such as those proposed by a newly trained, domain-specific tokenizer. Specifically, each new token is

[1] https://github.com/taidopurason/tokenizer-extension

tokenized with the original tokenizer and then recreated by merging the constituent tokens from left to right, adding intermediate merges and scaffold tokens. AdaptBPE (Balde et al., 2024) integrates new tokens via longest string matching before tokenization, but this alters the tokenization function itself.

In contrast to the aforementioned methods, rather than specifying the desired tokens in advance and trying to integrate them, **continued BPE training** (ours) directly continues the BPE training process from the pre-trained tokenizer on in-domain data. This ensures that all added tokens and merges are fully compatible and optimal under the BPE objective, without introducing any unreachable tokens, thereby solving both the problem of identifying the tokens to add and their optimal integration.

2.2 Tokenizer pruning

In addition to extension, we introduce improved pruning techniques that surpass naive frequency-based methods (Yang et al., 2022; Csaki et al., 2023), allowing for more extensive and targeted pruning without compromising compression or downstream performance. Prior pruning methods, such as BPE-Trimming (Cognetta et al., 2024), target low-frequency tokens and split them during inference. We extend this work by introducing leaf-based pruning and trimming the tokenizer itself so that the inference remains identical to the basic BPE. BPE-Knockout (Bauwens and Delobelle, 2024) removes tokens that have little value in terms of morphology and introduces tuple merges to compensate for their absence. PickyBPE (Chizhov et al., 2024) and Scaffold-BPE (Lian et al., 2025) represent a different line of work of refining the vocabulary during tokenizer training by removing the intermediate, "scaffolding" tokens. While our pruning method also removes tokens, it does so to eliminate unused tokens in application to a narrower set of languages or domains. Our approach therefore modifies only the vocabulary and merge list while leaving the tokenization function unchanged, maintaining full compatibility with the original implementation and enabling further extensions.

2.3 Replacing the tokenizer

Instead of pruning and extending, previous works have also successfully replaced the previous tokenizer with a new one entirely (Samuel et al., 2025). However, Dagan et al. (2024) show that replacing or significantly altering the tokenizer requires extensive continued pre-training to avoid performance degradation, and Tejaswi et al. (2024) find that the larger the vocabulary extension, the longer training is needed. We leave the question of what is the best strategy—modifying or replacing the tokenizer—to future research.

2.4 Modification of embeddings

When the tokenizer vocabulary is modified, the model embeddings must be updated accordingly. Tejaswi et al. (2024) and Csaki et al. (2024) find that Fast Vocabulary Transfer (FVT, Gee et al., 2022), which copies embeddings for overlapping tokens and initializes new ones by averaging those of their decompositions under the original tokenizer (see Appendix E), performs competitively among initialization strategies. While alternative methods exist (Dobler and de Melo, 2023; Samuel et al., 2025), our work focuses on selecting which tokens to add or prune at the tokenizer level rather than on embedding initialization.

3 Methodology

3.1 Vocabulary extension

Naive extension It is common to extend a tokenizer with new tokens by training a new tokenizer with in-domain data and adding the non-overlapping tokens in the order of the trained vocabulary. As HF transformers relies on merge lists, it is also necessary to create merges leading to the added tokens. We generate the merges leading to the new tokens based on the token priorities, as opposed to simply merging the two tokenizers' merge lists, since we found the prior provides better results (see Appendix B.5). We refer to this method as **the naive approach**.

Continued tokenizer training As an improvement, we propose **continued tokenizer training** to improve tokenization efficiency (see Figure 3). We define **continued BPE training** as reapplying the BPE tokenizer training algorithm to in-domain data, using token pair frequencies of the text tokenized with the existing tokenizer to create the new merges. During merge creation, we ensure that each resulting token is valid according to the training implementation of the underlying tokenizer.

For Byte-level BPE models (BBPE Radford et al., 2019), this involves first applying the pre-tokenizer to segment the text, then tokenizing the resulting segments to count token pair frequencies within the pre-tokens.

For SentencePiece BPE models (Kudo and Richardson, 2018), we tokenize the text and follow SentencePiece's merging rules by respecting constraints such as keeping different scripts separate and only allowing the space symbol at the beginning of a token. If necessary, we also extend the vocabulary with missing characters to ensure coverage before finding extension merges and ensure that the training data is Unicode NFKC normalized.

In both cases, we then apply the standard BPE algorithm for tokenizer training using pair frequencies from the tokenized texts and creating merges until the desired number of new tokens is reached. For extending the original tokenizer we simply append the new tokens to the vocabulary and the new merges to the merge list in the order they were created. Similar to BPE tokenizer training, this method is language-agnostic and only requires texts in the target language.

3.2 Vocabulary Pruning

To allow for controlled reduction of vocabulary size, we also implement vocabulary pruning. This process involves removing tokens and the merges that lead to them, freeing space for vocabulary extension. To determine the tokens that should be pruned first, we try several approaches, based either on token ID (Last N) or on token frequency in a representative text sample. When implemented naively, both approaches might lead to unreachable tokens: IDs do not always reflect the order of merges, as in Llama 3.2 (Grattafiori et al., 2024), and intermediate tokens tend to have lower frequencies than their downstream full-word tokens (Chizhov et al., 2024).

Addressing this issue, we implemented a **leaf-based pruning** algorithm, which iteratively collects the pruning order of tokens that are leaves in the BPE merge graph, i.e., they do not participate in merges and thus do not produce any downstream tokens. In Algorithm 1, we show pseudocode for frequency-based pruning of leaf tokens. First, atomic and leaf tokens are selected using the merge list. Leaf tokens are used to populate the priority queue, while atomic tokens cannot be removed and should never be added to the queue. After this, on each iteration, the lowest-priority element is taken from the queue and split into its merging sub-tokens. For each of the subtokens, we check the number of downstream merges it participates in, and, if there are no downstream merges left, this token is also considered a leaf and added to the priority queue for removal. The process ends when the queue runs out of leaves, which is the point when all non-atomic tokens are split.

Algorithm 1: Frequency Leaf Pruning

Input:
`vocab`: mapping "token → vocabulary index"
`unreachable`: set of unreachable tokens
`merges`: list of BPE merges
`counts`: mapping "token → frequency in the target corpus"
Output:
`prune_order`: list of tokens to prune in the order of pruning.

initialize `atomics` ← set of tokens not reachable by merges
initialize `leaves` ← set of merged tokens not participating in merges + `unreachable`
initialize `downstream_merges` ← mapping "token → number of merges with it"
initialize `token_splits` ← mapping "token → merge leading to it"
initialize `prune_order` ← []
initialize `queue` ← heap priority queue "leaf ← (frequency, index)"
while `queue` **:**
  `token` ← `queue`.popmin()
  `pruning_order`.append(`token`)
  (t_1, t_2) ← `token_splits`[`token`]
  update t_1, t_2 in `queue` by `queue`[`token`]
  for $t \in \{t_1, t_2\}$ **:**
    `downstream_merges`$[t]$ $-= 1$
    if (`downstream_merges`$[t] == 0$) & ($t \notin$ `atomics`) **:**
      add t to `leaves` and `queue`
return `pruning_order`

We also introduce a merge-based pruning algorithm, which yields results similar to leaf-based frequency pruning. As its performance closely matches that of leaf-based pruning, we provide a detailed description in Appendix A.

3.3 Detecting unreachable tokens

We extend the method of Land and Bartolo (2024) for identifying unreachable tokens to all vocabulary items (including those not decodable to UTF-8) and provide a formal definition. We define the **Self-Tokenization Test** (**STT**) as **the number of tokens unreachable through merges**:

$$STT = \sum_{t \in \mathcal{V}} \mathbb{1}\left[tokenize(t) \neq [t]\right]$$

The main idea is that if tokenizing a vocabulary token with the same tokenizer[2] does not reproduce

[2] In practice, this means that we test the raw string corresponding to the token, bypassing pre-processing, and disabling merge skipping (`ignore_merges`).

that token, then the token cannot be formed through merge operations[3].

Rationale. Let a BPE tokenizer be defined by a vocabulary $\mathcal{V}$ and a deterministic sequence of local merges, applied without merge skipping. For a token $t \in \mathcal{V}$, if tokenize$(t) \neq [t]$, then t is **unreachable** by merges. If tokenize$(t) = [t]$, then t could be reachable.

Suppose t appears in the tokenized output of some pre-tokenized input $\mathcal{I}$. Then there must exist a substring of $\mathcal{I}$ equal to the literal form of t that is merged into t. Because merges are local and merge skipping is disabled, the same sequence of merges must also apply when tokenizing the isolated string t. Therefore, if tokenize$(t) \neq [t]$, no input string can ever produce t, and t is unreachable. However, if tokenize$(t) = [t]$, then t is in principle reachable by merges assuming the pre-tokenizer produces t as a substring to some input.

Merge Skipping. If a pre-token output by a pre-tokenizer fully matches a vocabulary token, the merging process is skipped, and the token is produced immediately when merge skipping is enabled (`ignore_merges` in HF). When applying this method, merge skipping is disabled because otherwise any existing vocabulary token trivially returns $[t]$, making the test meaningless. Some tokenizers (e.g., Llama 3) rely on this mechanism to produce certain tokens that are reachable only via merge skipping. While our method offers a fast check of merge-based reachability, its interpretation must account for whether tokens are produced via merge skipping in practice.

4 Experimental setup

4.1 Datasets

We use Fineweb-2 (Penedo et al., 2025, ODC-By license, multilingual) and Fineweb (Penedo et al., 2024, ODC-By license, English) as the training datasets and set a training budget of 100M characters for the comparison across 70 languages (see Table 12 in Appendix F) and 1B characters for a more targeted evaluation for Estonian. We observe that increasing the amount of training data beyond this threshold does not yield significant improvements in tokenization efficiency during extension (see Appendix B.1).

We frame the extrinsic evaluation as a case study on Estonian, using English-Estonian datamix for both tokenizer and LM training. Our setup reflects a common bilingual LLM adaptation scenario: extending a model to a new language while preserving its English capabilities. To this end, we allocated a total budget of 24B characters, equally divided between Estonian and English. The corpus was then tokenized using the different tokenizers under evaluation, ensuring that the underlying data remained constant across experiments involving changes to the tokenizer.

4.2 Evaluation

We evaluate compression on the FLORES-200 (Costa-Jussà et al., 2022) `devtest` split using *bytes per token* ($BPT = \frac{\text{UTF-8 bytes}}{\text{tokens}}$) as the evaluation metric (higher is better). For comparison of different methods, we report the relative difference $\Delta_{rel}BPT = \frac{BPT_{ours}}{BPT_{naive}} - 1$, which essentially states how many additional tokens the naive method produces compared to ours, as the number of bytes remains constant ($\frac{tokens_{naive}}{tokens_{ours}} - 1$). We additionally report Rényi efficiency (Zouhar et al., 2023, higher is better), which has been shown to correlate with downstream performance.

For downstream evaluation in Estonian and English, we evaluate on FLORES-200 (NLLB et al., 2022) (*ET↔EN*), Winogrande (Sakaguchi et al., 2021), Winogrande-ET (Ojastu et al., 2025), XCOPA (Ponti et al., 2020), SIB200 (Adelani et al., 2024), and Belebele (Bandarkar et al., 2024) using `lm-evaluation-harness` (Gao et al., 2023). We evaluate FLORES-200 with COMET (`Unbabel/wmt22-comet-da`, Rei et al., 2020, 2022). See Appendix D for more details.

4.3 Models

We look at four models Llama-3 (Grattafiori et al., 2024), Llama-2 (Touvron et al., 2023), Qwen-2.5 (Qwen et al., 2025) and, Mistral Nemo[4]. While Llama-3, Qwen-2.5, and Mistral Nemo use the BBPE tokenizer, while Llama-2 uses a SentencePiece BPE tokenizer. We use the tokenizers through Hugging Face `transformers` implementation (Wolf et al., 2020). For experiments involving model training, we use Llama-3.2 1B and 3B and modify its embeddings after vocabulary modification with FVT (Gee et al., 2022).

[3] It may still be produced through merge skipping.

	Compression ↑		Rényi effic.↑		Unreach. added↓	
Model	Δ_{rel}	**WR**	Δ_{rel}	**WR**	**ours**	**naive**
Llama-3						
+1000	3.3%	88.6%	-0.1%	18.6%	0.0%	4.5%
+2000	3.6%	92.9%	-0.6%	4.3%	0.0%	5.7%
+4000	3.8%	98.6%	-1.1%	5.7%	0.0%	7.0%
+8000	3.7%	100.0%	-1.4%	1.4%	0.0%	8.2%
+16000	3.3%	98.6%	-1.3%	1.4%	0.0%	9.3%
+32000	2.7%	92.9%	-1.1%	7.1%	0.0%	10.2%
Qwen-2.5						
+1000	4.3%	98.6%	0.1%	18.6%	0.0%	4.3%
+2000	5.6%	97.1%	-1.0%	7.1%	0.0%	5.6%
+4000	6.9%	100.0%	-1.8%	2.9%	0.0%	7.1%
+8000	8.1%	98.6%	-2.5%	1.4%	0.0%	8.4%
+16000	9.0%	97.1%	-3.0%	1.4%	0.0%	9.6%
+32000	9.6%	94.3%	-3.2%	8.6%	0.0%	10.5%
Mistral Nemo						
+1000	3.0%	97.1%	-0.5%	12.9%	0.0%	3.8%
+2000	3.7%	94.3%	-0.8%	5.7%	0.0%	4.9%
+4000	4.2%	98.6%	-1.0%	4.3%	0.0%	6.3%
+8000	4.5%	98.6%	-1.1%	4.3%	0.0%	7.5%
+16000	4.6%	100.0%	-1.1%	4.3%	0.0%	8.7%
+32000	4.3%	98.6%	-0.9%	4.3%	0.0%	9.7%
Llama-2 (SentencePiece)						
+1000	4.6%	95.7%	-0.2%	17.1%	0.0%	3.6%
+2000	5.2%	95.7%	-1.1%	8.6%	0.0%	4.8%
+4000	6.1%	90.0%	-1.8%	5.7%	0.0%	6.0%
+8000	6.9%	84.3%	-2.3%	7.1%	0.0%	7.0%
+16000	7.4%	72.9%	-2.7%	17.1%	0.0%	7.8%
+32000	7.7%	74.3%	-2.8%	21.4%	0.0%	8.4%

Table 1: **Average scores across 70 languages**. We report the average relative gain (Δ_{rel}) and Win Rate (*WR*) of continued tokenizer training (ours) over extending from an independently trained tokenizer (naive) for compression (*BPT*) and Rényi efficiency. We also report the percentage of added tokens that are unreachable through merges (*STT*). *WR* – the percentage of languages where our method scores higher than the naive method.

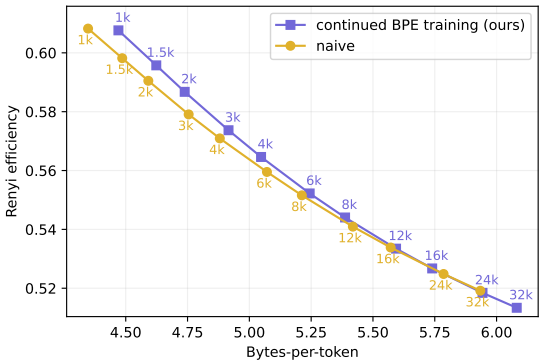

Figure 4: Llama-3 tokenizer Rényi efficiency ↑ vs compression (bytes per token) ↑ averaged over 70 languages. The point labels indicate the number of added tokens.

		Δ_{rel} **Compression↑**			
Script	(n)	**Llama-3**	**Qwen-2.5**	**Mistral**	**Llama-2**
Latn	(35)	4.7%	9.8%	3.3%	10.1%
Cyrl	(9)	5.1%	11.2%	5.4%	8.7%
Arab	(4)	0.7%	3.8%	0.4%	0.2%
Jpan	(1)	1.4%	1.1%	1.4%	-1.0%
Hani	(1)	-0.0%	0.2%	0.0%	-0.1%
Other	(20)	1.2%	2.3%	5.9%	0.5%

Table 2: Relative improvement (%) of continuing tokenizer training over extending from a tokenizer trained from scratch (naive) in *bytes per token*. We calculate the mean improvement across extension sizes 1k, 2k, 4k, 8k, 16k, 32k and 70 languages grouped by scripts.

5 Results

5.1 Tokenizer evaluation

By continuing tokenizer training (ours) instead of extending a tokenizer from a new language-specific tokenizer (naive method), **we achieve up to 9.6% higher tokenization efficiency on average**, with a vast majority (72.9%–100% depending on the tokenizer and extension size) of the 70 languages achieving higher compression (see Table 1). We do not observe any notable negative effects of our method on compression, while the increase can exceed 20% compared to the naive method (see Appendix G.2 for full results). Furthermore, there is also almost no effect on English tokenization after the target-language extension (see Table 6 in Appendix B.3). Higher target-language compression means we can process the same text with fewer tokens during training and inference, consuming less computational resources.

When looking at Rényi efficiency (higher is better), there is a deterioration when comparing continued BPE training to naive extension. On a closer look, even though Rényi efficiency decreases for the model with the same number of added tokens, when looking at extended tokenizers with roughly equal compression, our method outperforms the naive method on average (see Figure 4).

While our method is language-agnostic, we observe that the improvement over the naive method depends on the base tokenizer, language, and the number of added tokens. If we examine the average increase in compression of our method over the naive method, we see that it is more effective for languages with Cyrillic and Latin scripts (see Table 2).

Why is continued BPE training more effective? In both vocabulary extension methods, each newly added token has merges leading to it from the tokens already in the tokenizer. However, this does not guarantee that these tokens will actually be produced during inference. To quantify this, we introduce the Self-Tokenization Test (see Section 3.3), which calculates *tokens unreachable*

[4] mistralai/Mistral-Nemo-Base-2407

	Llama-3			Qwen-2.5		
Ext. size	**ours**	**naive**	**WR**	**ours**	**naive**	**WR**
+1000	0.0%	4.0%	85.7%	0.0%	8.3%	88.6%
+2000	0.1%	4.3%	92.9%	0.0%	9.7%	98.6%
+4000	0.1%	4.6%	98.6%	0.1%	11.0%	100.0%
+8000	0.5%	5.1%	98.6%	0.4%	12.6%	100.0%
+16000	2.1%	6.9%	100.0%	2.0%	15.3%	100.0%
+32000	7.1%	11.8%	100.0%	7.0%	20.9%	100.0%

Table 3: Unused added tokens in practice (↓, average across 70 languages). Percentage of the added tokens that were not produced when tokenizing 10,000 target-language documents sampled from Fineweb-2 test set.

through merges. **Extending from an independently trained tokenizer (naive) produces more unreachable tokens compared to the continued tokenizer training, which does not produce any** (see Table 1). This means that naive vocabulary extension has added tokens that will never be used in the merge process.

It is important to note that the Llama-3 and Mistral Nemo use merge skipping by default, so some of the tokens unreachable through merges will still be produced in practice. To quantify the effect of this, we disable merge skipping for Llama-3 and Mistral Nemo and find that for the naive method, the compression drops 1.2%–5.8% and 0.5%–4.1% respectively on average across 70 languages depending on the extension size (see Table 7 in Appendix B.4). For our method, the difference is less than 0.1% on average. This might explain why the difference between our method and the naive method is smaller for Llama-3 and Mistral Nemo, that use merge skipping, compared to Qwen-2.5 and Llama-2, which do not.

In addition to calculating the number of tokens we can not reach through merges, we also tokenized the held-out target-language dataset using extended Llama-3 and Qwen-2.5 tokenizers to see which tokens are not produced in practice. On average, across 70 languages, our method yields a 4%–13.9% improvement in the utilization of the added token vocabulary in practice (see Table 3). Furthermore, we observed more low-frequency tokens after naive extension compared to continued BPE training when visualizing Estonian token counts (see Appendix B.2).

Pruning. When pruning and then extending the tokenizers[5] (see Table 4), we see that in terms of text compression, frequency-aware methods lead to better progress in added language (Estonian) and slower degradation in the base tokenizer language (English). In addition, leaf-based methods perform comparably or better than analogous naive implementations, as they account for the BPE structure. This is more evident in the number of unreachable tokens, where naive frequency-based pruning leads to a dramatic increase in unreachable tokens, breaking the BPE merge paths. Our results conclude that the leaf-based frequency pruning is the most effective approach. Merge-based pruning, our other proposed algorithm, performs comparably to the leaf-based pruning.

We provide a comparison of bilingual pruning without extension for nine languages in Appendix G.1, similarly demonstrating the benefits of leaf- and merge-based pruning methods.

5.2 Downstream evaluation

5.2.1 Continued pre-training

Downstream task results for the continually pre-trained Llama-3.2 1B and 3B models in Table 5 show no substantial difference between the naive and continued tokenizer training approaches. In these experiments, we first applied leaf-based frequency pruning and then extended the vocabulary to maintain a constant size. When comparing the original and modified tokenizers, the 1B model performs similarly in both cases, while the 3B model exhibits slightly lower scores on EN-ET machine translation and Belebele (ET) for the modified tokenizers, while having similar results on other benchmarks. Nevertheless, modifying the tokenizer reduced total training time by 26% compared to the base tokenizer, owing to improved compression for Estonian (see Table 9). When looking at the embeddings, we observe signs of undertrained tokens for the naive method after continued pre-training compared to our method. A detailed analysis is provided in Appendix B.2.

5.2.2 Model Compression via Pruning

If we consider the scenario of using a model for a specific language or a domain, multilingual tokenizers typically include tokens for many languages that may not be required. We investigate how many of those tokens we can remove without affecting the downstream performance. Focusing on Estonian and English, we remove tokens from the vo-

[5]It is also possible to first extend and then prune; however, we did not observe any meaningful difference in performance depending on the order (see Appendix B.6).

	N_Δ	Continued tokenizer training (ours)					Naive extension				
		ID	ID (L*)	Freq	Freq (L*)	Merge	ID	ID (L*)	Freq	Freq (L*)	Merge
ET (BPT)	**1000**	3.289	3.289	3.289	3.289	3.289	3.164	3.164	3.164	3.164	3.164
	2000	3.526	3.526	3.527	3.527	3.527	3.362	3.362	3.362	3.362	3.362
	4000	3.786	3.786	3.788	3.788	3.788	3.592	3.592	3.595	3.595	3.595
	8000	4.109	4.109	4.110	4.110	4.110	3.870	3.870	3.874	3.874	3.874
	16000	4.451	4.452	4.456	4.456	4.456	4.202	4.202	4.208	4.208	4.208
	32000	4.829	4.829	4.837	4.837	4.837	4.552	4.552	4.567	4.567	4.566
	64000	5.164	5.164	5.197	5.198	5.198	4.907	4.907	4.943	4.943	4.943
	112000	5.381	5.381	5.407	5.386	5.386	5.147	5.147	4.732	5.154	5.174
EN (BPT)	**1000**	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862
	2000	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862
	4000	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862
	8000	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.862
	16000	4.862	4.862	4.862	4.862	4.862	4.862	4.862	4.863	4.863	4.863
	32000	4.852	4.852	4.863	4.863	4.863	4.853	4.853	4.864	4.864	4.864
	64000	4.740	4.740	4.869	4.869	4.868	4.738	4.738	4.868	4.868	4.868
	112000	4.219	4.219	4.402	4.385	4.387	4.206	4.206	4.377	4.382	4.384
Unreach. tokens	**1000**	556	551	541	534	534	630	625	615	608	608
	2000	536	527	509	488	504	710	701	683	662	678
	4000	483	465	446	386	445	834	817	798	738	797
	8000	371	337	371	176	328	1273	1239	1272	1077	1230
	16000	173	136	980	0	132	2196	2159	3015	2035	2167
	32000	0	0	2309	0	0	4656	4656	7002	4693	4693
	64000	0	0	2711	0	0	9581	9581	12663	9864	9868
	112000	0	0	2923	0	0	15279	15279	37637	16567	16813

Table 4: **Interaction of pruning and extension** (Llama-3). Bytes-per-token (BPT) on FLORES and unreachable tokens with different pruning and extension methods. In each experiment, we prune the tokenizer by N_Δ tokens using 50%–50% Estonian-English data mix, and then extend it back to the original size using Estonian-only data. *L indicates that leaf-based pruning was used.

	Compr.		FLORES (COMET)		Winogrande (acc)		Belebele (acc)		SIB200 (acc)		XCOPA (acc)
Model	**ET (EN)**	**Toks.**	**EN-ET**	**ET-EN**	**ET**	**EN**	**ET**	**EN**	**ET**	**EN**	**ET**
Llama-3.2-1B											
No training	2.64 (4.86)	0	0.432 ± 0.008	0.701 ± 0.007	50.0 ± 2.3	61.2 ± 2.7	28.6 ± 3.0	35.7 ± 3.1	71.6 ± 6.2	77.0 ± 5.8	51.0 ± 4.4
Continued pretraining:											
- default tokenizer	2.64 (4.86)	7.2B	0.788 ± 0.008	0.803 ± 0.005	58.1 ± 2.3	59.1 ± 2.7	28.0 ± 2.9	26.9 ± 2.9	77.9 ± 5.7	74.5 ± 6.0	63.6 ± 4.2
- prune+ext (16k, naive)	4.21 (4.86)	5.4B	0.790 ± 0.008	0.809 ± 0.005	57.4 ± 2.3	58.8 ± 2.7	25.6 ± 2.9	25.7 ± 2.9	73.0 ± 6.1	73.0 ± 6.1	62.6 ± 4.2
- prune+ext (16k, ours)	4.46 (4.86)	5.3B	0.796 ± 0.007	0.803 ± 0.005	58.3 ± 2.3	58.6 ± 2.7	27.3 ± 2.9	27.1 ± 2.9	76.5 ± 5.8	73.0 ± 6.1	63.0 ± 4.2
Llama-3.2-3B											
No training	2.64 (4.86)	0	0.635 ± 0.010	0.796 ± 0.006	53.0 ± 2.3	69.1 ± 2.5	44.1 ± 3.2	74.2 ± 2.9	74.5 ± 6.0	76.0 ± 5.9	55.4 ± 4.4
Continued pretraining:											
- default tokenizer	2.64 (4.86)	7.2B	0.835 ± 0.006	0.830 ± 0.005	65.1 ± 2.2	68.2 ± 2.6	52.9 ± 3.3	64.7 ± 3.1	80.9 ± 5.4	77.0 ± 5.8	70.4 ± 4.0
- prune+ext (16k, naive)	4.21 (4.86)	5.4B	0.818 ± 0.008	0.831 ± 0.005	63.2 ± 2.2	68.2 ± 2.6	51.1 ± 3.3	66.0 ± 3.1	81.4 ± 5.4	78.9 ± 5.6	69.8 ± 4.0
- prune+ext (16k, ours)	4.46 (4.86)	5.3B	0.822 ± 0.007	0.829 ± 0.005	63.1 ± 2.3	68.4 ± 2.6	48.7 ± 3.3	65.0 ± 3.1	78.4 ± 5.7	77.9 ± 5.7	71.8 ± 3.9

Table 5: Downstream task evaluation results on Llama-3 family of models after continued pre-training with English-Estonian data mix for different tokenizer setups (see Table 9 for tokenizer metrics).

cabulary and the embeddings while aiming to preserve performance in these two languages.

With our pruning methods, up to 62.5% (80k/128k) of the vocabulary tokens can be removed without any noticeable deterioration in downstream performance (see Figure 5). When comparing pruning strategies, leaf-based frequency pruning and merge-based pruning consistently allow for more extensive token removal than last-N pruning and naive frequency pruning. The latter two methods exhibit performance degradation earlier, whereas our approaches maintain robustness under higher pruning ratios. We also observe that machine translation (generative) is more sensitive to vocabulary pruning than the discriminative tasks, where the amount of tokens that can be removed without loss in performance is even higher than 62.5%. We also observe similar results for German-English pruning (see Figure 11 in Appendix G.1).

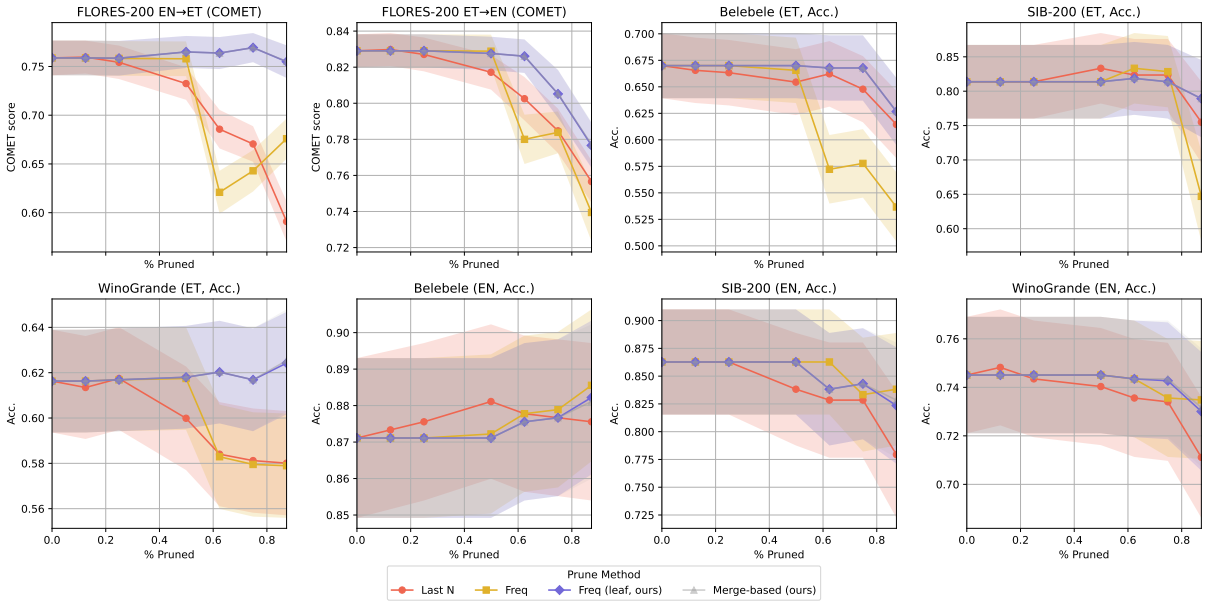

Figure 5: Llama-3.1-8B vocabulary pruned with different methods and evaluated on Estonian and English downstream tasks. The dataset for training the pruner consisted of 50-50 Estonian-English mix. The highlighted area shows the 95% confidence interval acquired through bootstrap resampling. Note that *Freq (leaf)* and *Merge-based* method results overlap.

6 Discussion

Our method of continued tokenizer training yields better intrinsic tokenizer performance than the commonly utilized naive extension method. More importantly, unlike naive extension, it does not harm the BPE structure and leaves no unreachable tokens. Unreachable tokens introduced by the naive method are only partially addressed by merge skipping, as merge skipping uses only the tokens that constitute complete pre-tokens.

Leaf-based pruning enables structure-aware tokenizer trimming, which can be utilized in conjunction with the tokenizer extension or independently to reduce model size, a factor that is now largely dependent on embedding parameters, particularly for small models. Furthermore, introducing frequencies into this pruning approach makes it possible to remove the vocabulary needed the least. For example, it is possible to combine a representative sample of all languages or domains required in the fine-tuned model, not only the ones directly participating in the tokenizer extension.

7 Conclusion

We presented methods for controlled vocabulary modification of pre-trained language models through improved tokenizer extension and pruning. Our proposed continued BPE training extends an existing tokenizer directly on domain-specific data, avoiding the inefficiencies of appending tokens from an independently trained tokenizer. Experiments across multiple languages and model families demonstrate that this approach enhances tokenization efficiency and more effectively utilizes the extended vocabulary. In parallel, our leaf-based pruning algorithm enables safe vocabulary reduction by removing redundant tokens without degrading model performance. Together, these techniques provide a practical framework for adapting tokenizers to new domains and languages while maintaining model compactness and quality. To help facilitate further research on vocabulary adaptation, we release an open-source toolkit implementing our methods, while preserving compatibility with the Hugging Face format.

Limitations

While our method demonstrates improved tokenization efficiency through continued BPE training, several limitations remain.

Our experiments were conducted in a bilingual setting, where a single language was added to an existing tokenizer. Although this setup is typical in low-resource adaptation, more complex multilingual scenarios remain unexplored and warrant future investigation.

The current approach is also restricted to BPE-

based tokenizers, which are still the predominant choice in large language models. Extending the method to other segmentation algorithms could provide additional insights.

Although our evaluation focuses on large language models, the method itself is not inherently limited to them. It could be applied to any pretrained model that relies on a BPE tokenizer. However, other architectures, such as encoder-decoder NMT models or encoder-only language models, were not included in our experiments.

We deliberately limit the scope of this work to tokenizer-level modification and do not explore how to achieve the best model-level downstream performance. This question likely depends on multiple interacting factors, such as dataset size, model capacity, and embedding initialization strategies, and thus requires a more extensive investigation. Consequently, we do not address when a tokenizer should be replaced entirely or how much tokenizer extension is optimal from the perspective of downstream performance.

Finally, the extrinsic evaluation on downstream tasks was, for the most part, limited to a single language and model, which may affect the generalizability of our findings to other settings.

Acknowledgments

This work was supported by the National Program for Estonian Language Technology Program (project EKTB104) funded by the Estonian Ministry of Education and Research. The authors also acknowledge the HPC resource allocation by Erlangen National High-Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in a joint project with the Center for Artificial Intelligence (CAIRO), THWS.

References

David Ifeoluwa Adelani, Hannah Liu, Xiaoyu Shen, Nikita Vassilyev, Jesujoba O. Alabi, Yanke Mao, Haonan Gao, and En-Shiun Annie Lee. 2024. SIB-200: A simple, inclusive, and big evaluation dataset for topic classification in 200+ languages and dialects. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 226–245, St. Julian's, Malta. Association for Computational Linguistics.

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

Gunjan Balde, Soumyadeep Roy, Mainack Mondal, and Niloy Ganguly. 2024. Adaptive BPE tokenization for enhanced vocabulary adaptation in finetuning pretrained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 14724–14733, Miami, Florida, USA. Association for Computational Linguistics.

Gunjan Balde, Soumyadeep Roy, Mainack Mondal, and Niloy Ganguly. 2025. Evaluation of LLMs in medical text summarization: The role of vocabulary adaptation in high OOV settings. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22989–23004, Vienna, Austria. Association for Computational Linguistics.

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 and virtual meeting. Association for Computational Linguistics.

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

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

Marco Cognetta, Tatsuya Hiraoka, Rico Sennrich, Yuval Pinter, and Naoaki Okazaki. 2024. An analysis of BPE vocabulary trimming in neural machine translation. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pages 48–50, Mexico City, Mexico. Association for Computational Linguistics.

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

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.

Zoltan Csaki, Pian Pawakapan, Urmish Thakker, and Qiantong Xu. 2023. Efficiently adapting pretrained language models to new languages. *Preprint*, arXiv:2311.05741.

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.

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*.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation.

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.

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, Gaël Liu, and 196 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

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

Olivier Lacombe, Kathleen Kenealy, Kat Black, Ravin Kumar, Francesco Visin, and Jiageng Zhang. 2025. Introducing gemma 3 270m: The compact model for hyper-efficient ai. Google Developers Blog. Accessed: 2026-01-16.

Sander Land and Max Bartolo. 2024. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11631–11646.

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

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.

Team NLLB, Marta Ruiz Costa-jussà, James Cross, Onur cCelebi, Maha Elbayad, Kenneth Heafield,

Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Alison Youngblood, Bapi Akula, Loïc Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, and 20 others. 2022. No language left behind: Scaling human-centered machine translation. *ArXiv*, abs/2207.04672.

Marii Ojastu, Hele-Andra Kuulmets, Aleksei Dorkin, Marika Borovikova, Dage Särg, and Kairit Sirts. 2025. Estonian WinoGrande Dataset: Comparative Analysis of LLM Performance on Human and Machine Translation. *Preprint*, arXiv:2511.17290.

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*.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pages 30811–30849. Curran Associates, Inc.

Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*, volume 36, pages 36963–36990. Curran Associates, Inc.

Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. 2020. XCOPA: A multilingual dataset for causal commonsense reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2362–2376, Online. Association for Computational Linguistics.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. 2022. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A neural framework for MT evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

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.

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.

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.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System*

Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

Atsuki Yamaguchi, Aline Villavicencio, and Nikolaos Aletras. 2024. How can we effectively expand the vocabulary of llms with 0.01gb of target language text? *Preprint*, arXiv:2406.11477.

Ziqing Yang, Yiming Cui, and Zhigang Chen. 2022. TextPruner: A model pruning toolkit for pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 35–43, Dublin, Ireland. Association for Computational Linguistics.

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

A Merge-based pruning

Algorithm 2: Merge-Based Pruning (MBP)

Input:
`vocab`: mapping from token → vocabulary index
`tok_counts`: token frequencies in tokenized corpus
`merge_counts`: frequencies of merges performed when tokenizing the corpus

Output:
`prune_order`: list of tokens to prune in the order of pruning.

initialize `counts` ← `tok_counts`

for (t_1, t_2), n *in* `merge_counts` **:**
  `counts`$[t_1]$ ← `counts`$[t_1] + n$
  `counts`$[t_2]$ ← `counts`$[t_2] + n$

return **sort** tokens by ascending `counts`, break ties by descending length of the token and finally `vocab` index

As an alternative to the frequency based pruning with leaf ordering, we also developed the merge-based pruning (see Algorithm 2), which achieves similar results. The main idea is to count the token frequency throuout the mergining process. So, in practice this means counting the final token frequency and any time it appears in a merge, creating another token. Consequently, a vocabulary token will have a count of n, while any token it was merged from will have a count of at least n. This follows from the fact that standard BPE defines a single deterministic merge tree for each token, and tokens appearing earlier in such trees often participate in multiple merge trees. This means that if we sort by that count, breaking ties by token length (when a token and its predecessor in the merge tree have equal count) and token index, only the leaf nodes will be removed.

Since the result is very similar we decided to focus on the leaf-frequency pruning in the main part of the paper.

B Additional results

B.1 Training dataset size

The effect of the training set diminishes at higher sizes. We see minimal difference between 1B and 10B character budget in scores when extending the tokenizer (see Figure 6). Extending a tokenizer using continued training with 10M character training set yields a bigger effect than extending from a new tokenizer (naive) using 10B character training set – the choice of extension method is more important than the training budget at this scale.

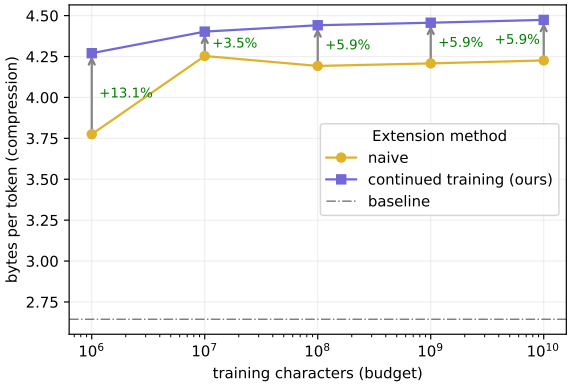

Figure 6: Extension training budget in characters vs compression of Estonian text when adding 16,000 Estonian tokens to the Llama-3 tokenizer.

B.2 Analysis of undertrained tokens

Frequency distribution of the added tokens is visualized in Figure 7. We observe that the naive method yields more low-frequency tokens than our continued BPE training method.

To better understand how this affects the learning of new tokens, we analyze the changes in their embeddings during continued pre-training. Since the added tokens are initialized with FVT and updated during training, undertraining is not always immediately visible. To investigate this, we measure both the L2 distance of embeddings from their initialized values and the change in their L2 norms. Figure 8 shows that the naive extension method produces noticeably more undertrained tokens, reflected in

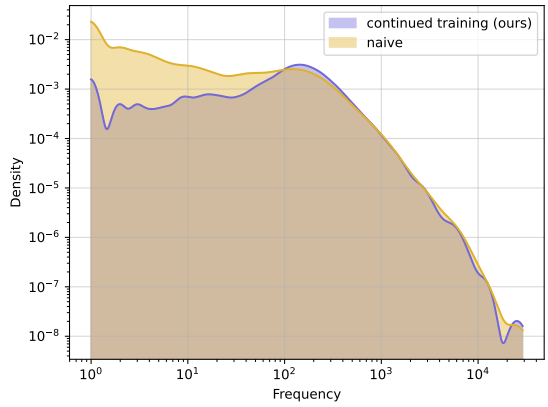

Figure 7: Token frequency distribution in the added tokens ($n = 8000$) in Llama-3 tokenizer on the held-out set of Estonian Fineweb-2 (10000 documents).

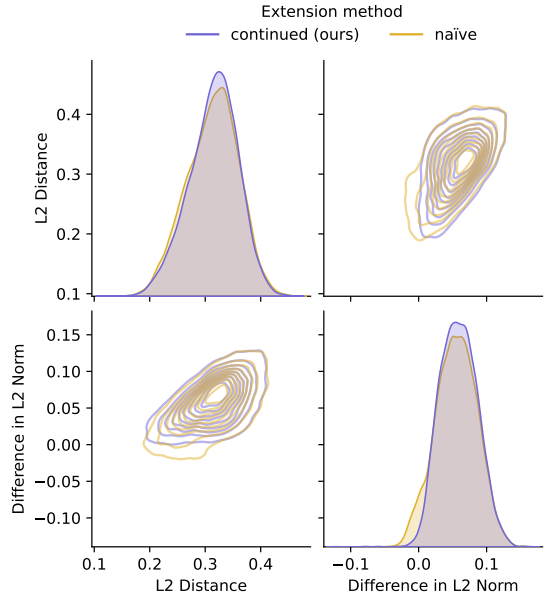

Figure 8: Comparison of input/output embeddings (tied) of added tokens at FVT initialization and after continued pre-training (L2 distance between them and the difference in L2 norms). We extend Llama-3.2-3B with Estonian tokens and continually pre-train on Estonian-English mix, comparing naive tokenizer extension and continued BPE training.

smaller L2 changes and a stronger weight-decay effect that drives their embeddings toward zero (visible as a longer tail of negative L2 norm differences). This interpretation is further supported by Table 3, which shows lower vocabulary utilization for the naive method.

B.3 Extension effect on English

Table 6 demonstrates that the target-language extension both with continued BPE training (ours) and the naive method has almost no effect on the tokenization of English sentences.

	Lev. dist. ↓		EM (%) ↑	
Ext. size	**ours**	**naive**	**ours**	**naive**
1000	0.001 <0.010	0.001 <0.010	100.0	99.9
2000	0.002 <0.012	0.002 <0.013	99.9	99.9
4000	0.003 <0.022	0.004 <0.022	99.9	99.8
8000	0.009 <0.036	0.010 <0.036	99.6	99.6
16000	0.023 <0.106	0.024 <0.098	99.1	99.0
32000	0.057 <0.198	0.057 <0.220	97.7	97.6

Table 6: Mean sentence-averaged Levenstein distance between the base tokenizer (Llama-3) and extended tokenizer tokenization of **English** FLORES sentences and the amount of sentences (%) where the tokenizations are an exact match (EM). We report the mean over 70 languages and additionally the worst-case language Lev. dist ($< \max$).

B.4 The effect of merge skipping

We note that merge skipping contributes significantly to the compression. We look at models that use merge skipping by default (Llama-3 and Mistral Nemo) and disable merge skipping to see how much the performance drops. In Table 7, see that for continued BPE training, disabling merge skipping has no significant effect, while for the naive method, the drop in tokenization efficiency is notable. This is because merge skipping allows reaching tokens that are unreachable by merges if those tokens are equal to a pre-token, bypassing merges. As we previously noted, the naive method produces many of those unreachable tokens.

	Llama-3		Mistral Nemo	
	ours	**naive**	**ours**	**naive**
+1000	-0.1%	-1.2%	0.0%	-0.5%
+2000	-0.1%	-1.9%	0.0%	-0.9%
+4000	-0.1%	-2.8%	0.0%	-1.6%
+8000	-0.1%	-3.8%	0.0%	-2.4%
+16000	-0.1%	-4.9%	0.0%	-3.3%
+32000	-0.1%	-5.8%	0.0%	-4.1%

Table 7: Change in compression (bytes per token) after disabling merge skipping depending on the tokenizer extension method. We report average change in performance across 70 languages for two tokenizers that use merge skipping.

B.5 Naive extension methods

There are multiple ways of adding tokens from an auxiliary independently trained tokenizer to an existing tokenizer (naive method).

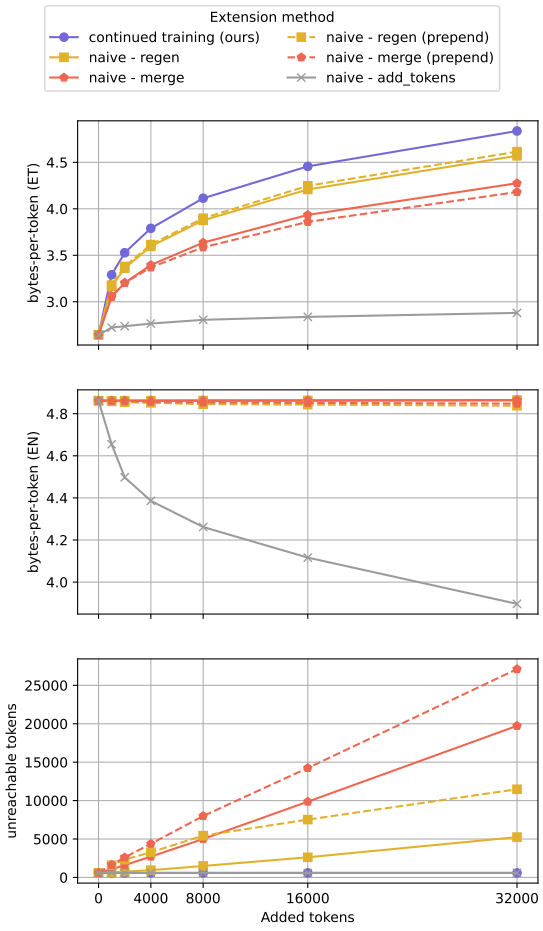

Figure 9: Comparison of **compression** (*bytes per tokens*↑) of Estonian (target language) and English, and **unreachable tokens**↓ given the number of added tokens for various methods of adding new tokens for Llama-3.

For tokenizers that do not define merge lists explicitly (e.g. SentencePiece), it is common to just append the vocabulary (Csaki et al., 2024; Cui et al., 2024; Tejaswi et al., 2024). For Hugging Face (HF) transformers tokenizer implementations, it is more complicated because merges are performed according to a merge list, so in addition to adding tokens, we also need to handle the merges. Some works have just appended the merge list of a new tokenizer (Csaki et al., 2023; Tejaswi et al., 2024), however, we show later that this is not optimal.

We took the approach of generating merges for the added vocabulary (*regen*) by checking which tokens can create the added token and then ordering the merge rules by the priority/index. Our implementation is based on the way that HF transformers converts SentencePiece tokenizers to their format, which requires creating merge lists. This is also similar to what happens when tokens are added to a SentencePiece tokenizer, since there are no merge lists that are merged together – the added tokens are produced through merging according to their priority.

In Figure 9, we observe that the approach of generating merge rules for the added tokens (*regen*) works better than appending merge rules (*merge*). We also compared against using the `add_tokens` method of HF transformers, however we find it has the lowest performance.

There are also works that have prepended merges (Csaki et al., 2023). When used with the regen method, it does result in higher scores, however we see that it affects the English tokenization and results in more unreachable tokens, so we do not use it.

Thus we decide to use the *regen* method, since we try to use the strongest baseline to compare our method against.

B.6 Order of pruning/extension

Table 8 shows that the difference between pruning and extending (in that order) vs extending and pruning is negligible in terms of compression.

	Prune-Extend		**Extend-prune**	
N_Δ	ET	EN	ET	EN
16000	4.441	4.863	4.441	4.863
32000	4.441	4.863	4.441	4.863
64000	4.440	4.861	4.439	4.861
96000	4.402	4.671	4.401	4.671

Table 8: Order of pruning and extension. Compression (bytes per token) for Llama-3 extension (Estonian) and pruning (Estonian-English). N_Δ is the number of tokens pruned and then extended (or vice versa).

C Continued pre-training details

We provide details for the modified Llama-3 tokenizers in the continued pertaining experiments in Table 9.

The hyperparamters for continued pre-training using Hugging Face `transformers` (Wolf et al., 2020) are in Table 10. The training was conducted on the LUMI supercomputer using 8 nodes, each consisting of 4 AMD MI250x GPUs (acting as 8 compute units). The continued pre-training experiments took 2311 GPU-hours (180–612 GPU-hours per experiment).

D Evaluation details

We provide additional evaluation dataset details in Table 11. The evaluation prompt setup is de-

Tokenizer	|vocab|	unreach	Compr. ET	Compr. EN	CPT tokens	% ET
default	128k	588	2.64	4.86	7.2B	63.9%
prune+ext (16k)	128k	0	4.46	4.86	5.3B	50.6%
prune+ext (16k, naive)	128k	2033	4.21	4.86	5.4B	52.0%

Table 9: Continued pertaining (CPT) tokenizer overview.

Hyperparameter	Value
Learning rate	4e-4 (1B) / 1e-4 (3B)
Optimizer	AdamW
Adam ϵ	1e-8
Adam β_1, β_2	0.9, 0.95
Sequence length	4096
Weight decay	0.1
Scheduler	cosine decayed to 10%
Warmup ratio	10%
FSDP Strategy	`SHARD_GRAD_OP`
GPUs	64
Precision	`bfloat16`
Batch size	1024
Batch size (tokens)	4194304

Table 10: Continued pre-training hyperparameters.

tailed in Figure 10. For discriminative benchmarks, we report accuracy and 95% confidence interval calculated from the standard error reported by `lm-eval-harness`. For FLORES-200 (MT) evaluation we calculate the confidence intervals using bootstrap resampling.

E Fast Vocabulary Transfer (FVT)

To formally define Fast Vocabulary Transfer (FVT Gee et al., 2022), let V and tok denote the vocabulary and tokenization function of the original tokenizer, and let $E(t)$ denote the embedding of token t. For a new tokenizer with vocabulary V', the embedding $E'(t)$ for each $t \in V'$ is defined as:

$$E'(t) = \begin{cases} E(t) & \text{if } t \in V, \\ \dfrac{\sum_{t_i \in tok(t)} E(t_i)}{|tok(t)|} & \text{otherwise.} \end{cases} \quad (1)$$

FLORES (*generative*)
```
Translate the text from {src_lang} to {tgt_lang}. \n
\n
English: {src_sent}\n
Estonian:
```

Belebele (*log-likelihood*)
```
P: {passage}\n
Q: {question}\n
A: {answer1}\n
B: {answer2}\n
C: {answer3}\n
D: {answer4}\n
Answer: {correct letter}
```

SIB-200 (*log-likelihood*)
```
Topic Classification: science/technology, travel, politics, sports, health, entertainment, geography.\n
\n
The topic of "[{text}]" is: {topic}
```

Winogrande (*log-likelihood*)
```
{pre_blank} {choice_1/choice_2} {post_blank}
```

X-COPA (*log-likelihood*)
```
{premise} {connector} {choice_1/choice_2}
```

Figure 10: Prompts used for evaluation with `lm-eval-harness` (Gao et al., 2023).

Dataset		Size	Split	n-shots	License
Winogrande	(Sakaguchi et al., 2021)	1267	valid	0	CC-BY
Winogrande ET	(Ojastu et al., 2025)	1767	test	0	Apache 2.0
XCOPA	(Ponti et al., 2020)	600	test	5	CC-BY-4.0
SIB200	(Adelani et al., 2024)	204	test	5	CC-BY-SA-4.0
FLORES200	(NLLB et al., 2022)	1012	devtest	5	CC-BY-SA-4.0
Belebele	(Bandarkar et al., 2024)	900	test	5	CC-BY-SA-4.0

Table 11: Evaluation datasets. *n-shot* is the number of in-context few-shot examples used for prompting during evaluation.

F Languages in training data

We provide the overview of the languages in the training dataset in Table 12.

Language	Code	Family	Bytes	Language	Code	Family	Bytes
Russian	rus_Cyrl	Indo-European	6.4T	Standard Estonian	ekk_Latn	Uralic	43.8B
Mandarin Chinese	cmn_Hani	Sino-Tibetan	2.7T	Croatian	hrv_Latn	Indo-European	38.6B
German	deu_Latn	Indo-European	1.7T	Standard Latvian	lvs_Latn	Indo-European	35.8B
Japanese	jpn_Jpan	Japonic	1.7T	Standard Malay	zsm_Latn	Austronesian	34.3B
Spanish	spa_Latn	Indo-European	1.4T	North Azerbaijani	azj_Latn	Turkic	28.9B
French	fra_Latn	Indo-European	1.2T	Tamil	tam_Taml	Dravidian	39.7B
Italian	ita_Latn	Indo-European	793.8B	Serbian	srp_Cyrl	Indo-European	28.8B
Portuguese	por_Latn	Indo-European	611.2B	Tosk Albanian	als_Latn	Indo-European	19.5B
Polish	pol_Latn	Indo-European	463.9B	Urdu	urd_Arab	Indo-European	21.4B
Dutch	nld_Latn	Indo-European	426.8B	Kazakh	kaz_Cyrl	Turkic	22.2B
Indonesian	ind_Latn	Austronesian	374.4B	Georgian	kat_Geor	Kartvelian	27.1B
Turkish	tur_Latn	Turkic	305.5B	Nepali (individual language)	npi_Deva	Indo-European	27.0B
Vietnamese	vie_Latn	Austro-Asiatic	343.4B	Marathi	mar_Deva	Indo-European	24.2B
Czech	ces_Latn	Indo-European	221.5B	Malayalam	mal_Mlym	Dravidian	23.9B
Standard Arabic	arb_Arab	Afro-Asiatic	315.2B	Macedonian	mkd_Cyrl	Indo-European	16.1B
Korean	kor_Hang	Koreanic	229.2B	Icelandic	isl_Latn	Indo-European	11.0B
Persian	fas_Arab	Indo-European	327.1B	Belarusian	bel_Cyrl	Indo-European	12.3B
Hungarian	hun_Latn	Uralic	214.4B	Telugu	tel_Telu	Dravidian	15.5B
Swedish	swe_Latn	Indo-European	217.9B	Afrikaans	afr_Latn	Indo-European	8.3B
Romanian	ron_Latn	Indo-European	199.9B	Kannada	kan_Knda	Dravidian	13.9B
Ukrainian	ukr_Cyrl	Indo-European	273.7B	Gujarati	guj_Gujr	Indo-European	12.6B
Norwegian Bokmål	nob_Latn	Indo-European	184.7B	Galician	glg_Latn	Indo-European	6.9B
Modern Greek (1453-)	ell_Grek	Indo-European	238.4B	Burmese	mya_Mymr	Sino-Tibetan	13.3B
Thai	tha_Thai	Kra-Dai	299.2B	Moroccan Arabic	ary_Arab	Afro-Asiatic	8.3B
Danish	dan_Latn	Indo-European	161.8B	Halh Mongolian	khk_Cyrl	Mongolic	9.1B
Finnish	fin_Latn	Uralic	153.6B	Armenian	hye_Armn	Indo-European	7.7B
Bulgarian	bul_Cyrl	Indo-European	156.5B	Khmer	khm_Khmr	Austro-Asiatic	9.3B
Slovak	slk_Latn	Indo-European	91.7B	Northern Uzbek	uzn_Latn	Turkic	4.8B
Hindi	hin_Deva	Indo-European	129.9B	Basque	eus_Latn	Language isolate	4.6B
Lithuanian	lit_Latn	Indo-European	60.7B	Sinhala	sin_Sinh	Indo-European	7.6B
Bosnian	bos_Latn	Indo-European	52.8B	Panjabi	pan_Guru	Indo-European	6.1B
Hebrew	heb_Hebr	Afro-Asiatic	73.8B	Kirghiz	kir_Cyrl	Turkic	4.7B
Bengali	ben_Beng	Indo-European	93.5B	Swahili (individual language)	swh_Latn	Niger-Congo	3.3B
Slovenian	slv_Latn	Indo-European	44.9B	Norwegian Nynorsk	nno_Latn	Indo-European	2.9B
Catalan	cat_Latn	Indo-European	43.3B	Odia	ory_Orya	Indo-European	5.3B

Table 12: Overview of the 70 languages used. We report the number of UTF-8 bytes in Fineweb-2 (Penedo et al., 2025).

G Full results

G.1 Extended tokenizer pruning results

We provide pruning compression and unreachable token curves for additional languages in Figures 12 and 13, respectively. Additionally, we provide downstream evaluation of pruning for German in Figure 11, where our pruning methods perform the best.

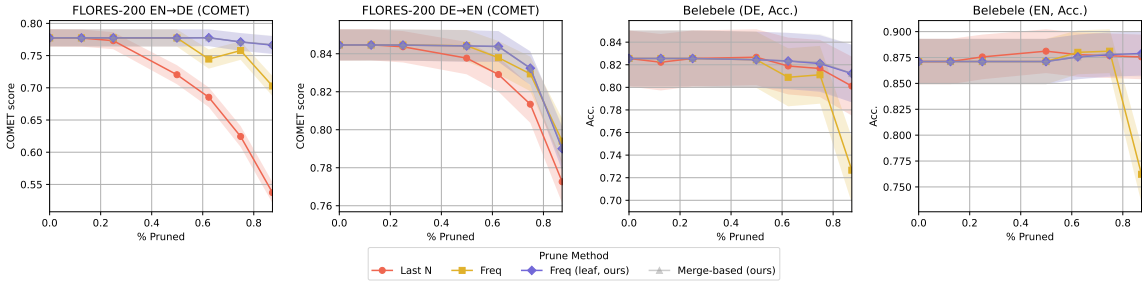

Figure 11: Llama-3.1-8B vocabulary pruned with different methods and evaluated on German and English downstream tasks. The dataset for training the pruner consisted of 50-50 German-English mix. The highlighted area shows the 95% confidence interval. Note that *Freq (leaf)* and *Merge-based* method results overlap.

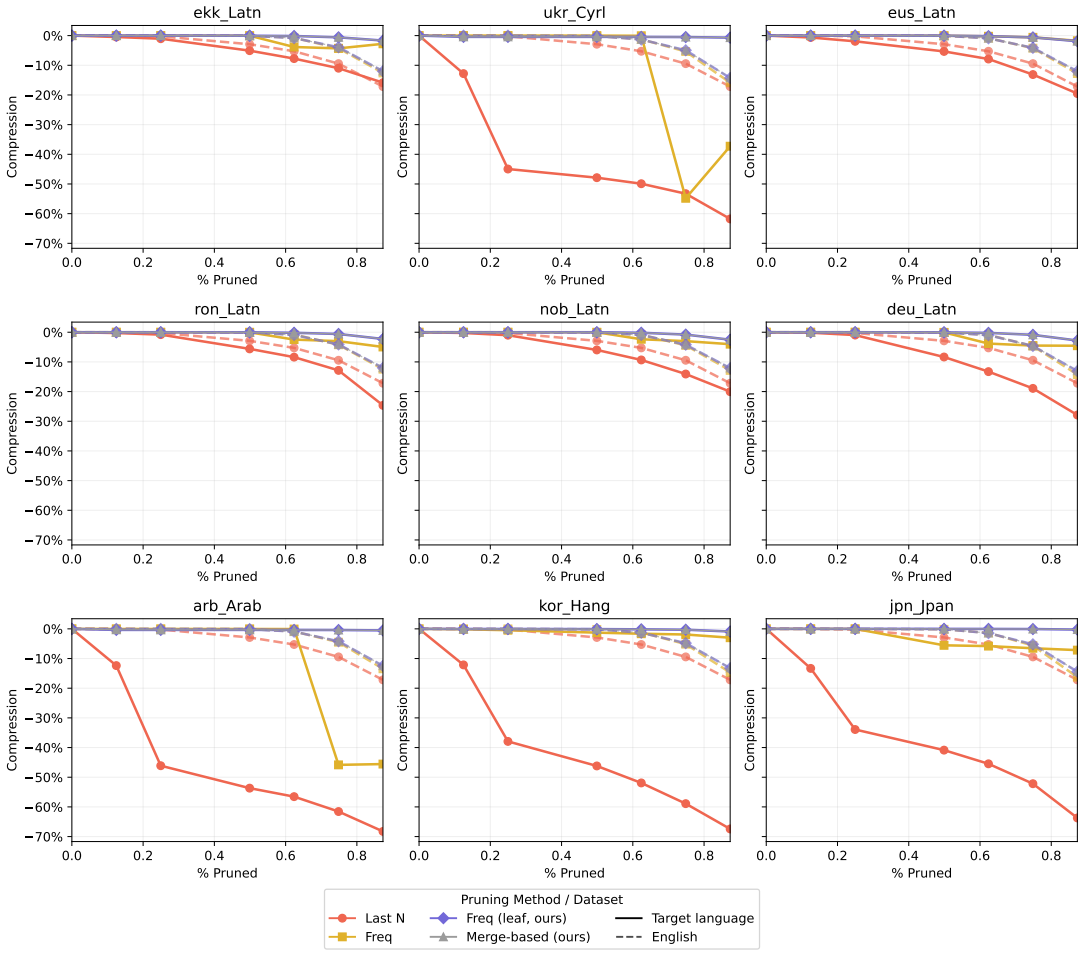

Figure 12: Change in **compression** (*bytes per token* ↑) depending on the pruning method. Note that *Freq (leaf)* and *Merge-based* method results overlap.

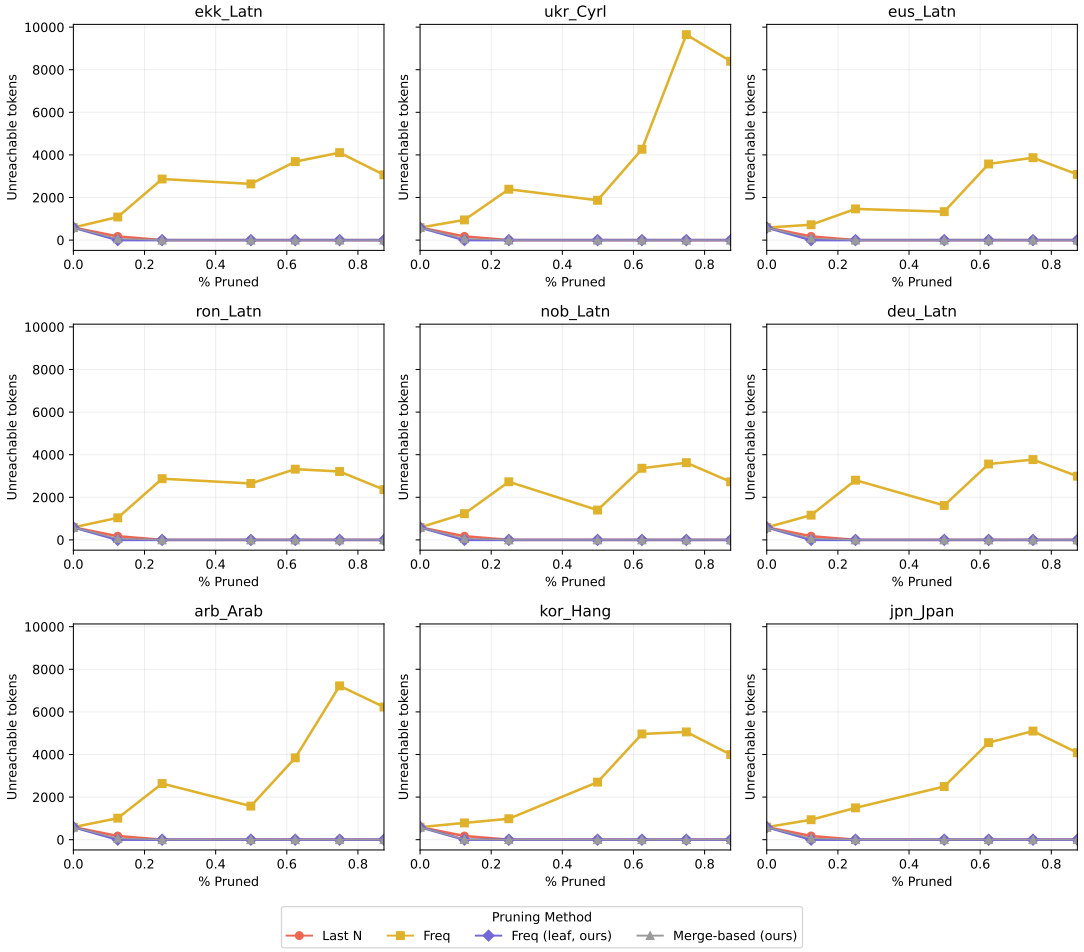

Figure 13: Number of **unreachable tokens**↓ created by different pruning methods.

G.2 Full tokenizer extension results

We provide the full improvement of our method (continued BPE training) over the naive method in Tables 13 and 14 for compression and in Table 15 and 16 for unreachable tokens. We list full compression values for continued BPE training in Tables 17 and 18.

	Llama-3 extension						Qwen-2.5 extension					
Lang.	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	1.9	2.1	2.3	1.7	1.4	1.2	3.4	4.4	5.6	6.6	7.2	7.2
als_Latn	5.0	4.6	4.1	3.9	2.6	1.9	8.0	9.5	10.8	11.0	10.0	10.1
arb_Arab	0.9	1.0	1.1	1.3	0.9	0.6	1.5	2.0	2.8	3.2	3.5	3.8
ary_Arab	0.4	0.9	0.9	0.9	0.9	0.6	0.6	1.3	1.5	1.7	2.3	2.4
azj_Latn	8.5	9.9	10.6	9.8	9.3	7.4	11.1	13.6	17.7	19.5	22.7	23.9
bel_Cyrl	3.7	4.5	4.8	4.6	3.6	2.6	3.8	5.2	6.7	8.4	9.0	9.4
ben_Beng	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
bos_Latn	7.5	9.5	9.9	10.2	9.2	8.1	9.1	11.8	15.4	19.1	22.9	26.8
bul_Cyrl	7.8	8.8	9.7	10.1	9.0	7.4	13.0	16.3	20.5	24.2	27.6	29.1
cat_Latn	2.7	2.8	2.5	2.2	1.7	1.0	3.6	4.3	4.9	5.2	5.7	5.6
ces_Latn	3.4	4.3	4.3	5.3	5.5	5.0	5.8	8.2	10.0	12.0	14.2	15.6
cmn_Hani	-0.0	-0.0	0.1	0.0	-0.0	-0.1	0.0	0.2	0.2	0.3	0.2	0.3
dan_Latn	2.1	2.3	2.3	2.3	2.1	1.8	2.8	3.6	4.4	5.2	5.3	5.8
deu_Latn	2.2	2.8	2.9	2.7	2.6	2.3	2.3	3.1	4.1	5.0	6.1	6.9
ekk_Latn	4.1	4.8	5.5	6.0	5.9	5.7	5.4	6.7	8.2	9.6	11.1	12.5
ell_Grek	1.5	1.6	1.9	1.9	1.3	0.7	0.1	0.2	0.3	0.3	0.4	0.0
eus_Latn	6.1	7.2	8.0	8.1	7.1	6.2	10.4	13.7	17.4	20.1	21.2	23.6
fas_Arab	0.7	0.6	0.5	0.2	0.2	0.1	3.2	4.4	5.3	6.4	5.4	4.2
fin_Latn	6.9	8.6	9.7	10.6	11.2	10.9	7.7	10.6	13.0	16.1	19.3	21.7
fra_Latn	2.1	2.1	2.0	1.7	1.4	0.8	2.3	2.5	2.8	3.5	4.2	4.0
glg_Latn	1.9	2.0	2.2	1.8	1.5	1.0	2.4	2.8	3.5	3.7	4.3	4.4
guj_Gujr	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0
heb_Hebr	29.5	27.1	23.7	19.6	14.7	9.1	9.8	14.1	17.9	21.3	24.7	27.3
hin_Deva	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hrv_Latn	8.3	9.4	11.1	11.4	10.6	8.5	9.7	13.1	17.4	21.8	26.2	30.3
hun_Latn	4.1	5.2	6.2	7.0	7.1	6.7	5.1	7.2	9.3	12.3	15.0	17.0
hye_Armn	0.0	0.0	-0.0	0.0	0.0	0.1	1.3	0.7	0.7	0.8	0.9	0.5
ind_Latn	2.3	2.1	2.3	1.6	0.9	0.6	4.7	6.2	8.2	9.0	9.3	8.6
isl_Latn	2.5	3.0	3.5	3.5	4.2	4.0	2.9	4.2	5.5	6.3	8.2	8.8
ita_Latn	5.4	5.5	5.7	4.7	3.3	2.5	5.9	7.3	9.0	10.7	12.3	13.7
jpn_Jpan	1.3	1.2	1.5	1.4	1.7	1.2	0.9	0.9	1.2	1.1	1.2	1.0
kan_Knda	-0.0	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.1	-0.1	0.0
kat_Geor	0.2	0.4	0.2	0.2	0.3	0.2	1.9	2.4	2.3	2.8	3.5	3.4
kaz_Cyrl	5.4	6.7	6.7	6.1	5.0	3.5	7.6	10.5	12.4	14.4	15.5	16.2
khk_Cyrl	3.8	5.5	5.1	3.8	3.0	2.3	7.1	8.9	11.8	12.0	11.5	10.4
khm_Khmr	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
kir_Cyrl	6.0	6.6	7.4	6.4	5.4	4.1	6.5	8.3	12.1	13.3	13.6	14.1
kor_Hang	0.4	0.5	0.6	0.7	0.8	0.8	0.6	0.8	0.9	1.0	1.2	1.3
lit_Latn	4.4	5.1	5.6	5.9	5.6	4.7	4.8	6.9	9.0	11.9	14.4	16.0
lvs_Latn	4.0	5.0	4.8	5.1	4.6	3.7	5.2	7.4	8.3	9.6	11.2	12.1
mal_Mlym	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.1	0.0	0.0
mar_Deva	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0
mkd_Cyrl	8.1	9.7	10.7	10.0	8.5	6.4	12.4	16.6	20.9	24.4	28.3	29.3
mya_Mymr	-0.1	0.0	0.0	0.0	0.0	0.0	0.5	0.6	0.6	0.6	0.6	0.6
nld_Latn	2.0	2.2	2.2	2.2	1.7	1.3	3.6	4.4	5.6	6.9	7.3	7.5
nno_Latn	1.9	2.2	2.5	2.3	2.2	1.8	3.6	4.4	4.1	4.4	4.9	5.4
nob_Latn	1.5	1.7	2.0	1.9	1.6	1.7	2.2	2.7	3.8	4.2	4.7	5.3
npi_Deva	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0
ory_Orya	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.2	0.2	0.0	0.0	-0.1
pan_Guru	0.0	0.2	0.2	0.2	0.2	0.2	-0.0	-0.0	0.0	0.0	0.0	0.0
pol_Latn	6.5	9.0	11.0	11.6	10.8	9.0	6.2	9.4	12.8	16.3	19.8	23.5
por_Latn	1.4	1.8	1.7	1.4	1.2	0.9	1.6	2.3	2.7	3.1	3.8	3.8
ron_Latn	5.5	5.2	5.6	5.3	4.4	3.1	6.6	7.6	9.0	10.2	11.2	11.2
rus_Cyrl	2.3	2.2	2.5	2.7	2.6	2.2	2.0	2.4	2.9	3.6	4.1	4.6
sin_Sinh	0.9	0.5	0.2	0.1	0.1	0.0	18.4	19.3	19.9	20.0	20.2	20.0
slk_Latn	5.9	7.0	7.1	7.6	7.2	6.5	6.3	9.3	12.1	14.1	17.2	18.6
slv_Latn	7.0	8.2	9.1	9.4	8.6	7.2	8.6	10.8	13.7	16.8	19.6	22.0
spa_Latn	2.9	2.9	2.5	2.2	1.6	1.1	3.0	3.2	3.3	4.0	4.5	4.7
srp_Cyrl	2.9	3.6	4.4	4.0	3.6	2.8	3.1	4.8	6.1	7.6	9.1	9.8
swe_Latn	1.6	1.9	2.1	2.3	2.1	1.9	2.4	3.0	3.8	4.8	5.5	5.9
swh_Latn	9.2	10.9	10.9	10.9	9.4	7.5	16.4	23.2	30.2	36.2	39.9	40.8
tam_Taml	0.0	0.0	0.0	0.0	0.0	-0.0	0.0	0.0	0.0	0.0	0.0	-0.0
tel_Telu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0
tha_Thai	0.4	0.4	0.6	0.4	0.4	0.5	1.1	0.7	1.1	1.1	1.1	1.1
tur_Latn	5.0	5.3	5.6	6.0	5.7	4.6	3.8	5.5	7.5	9.1	10.7	11.8
ukr_Cyrl	2.5	2.4	2.7	2.8	2.6	2.2	2.9	3.9	4.9	6.1	6.9	7.4
urd_Arab	1.2	0.8	0.4	0.5	0.3	0.2	5.7	6.0	7.3	7.3	5.0	3.3
uzn_Latn	9.8	11.3	11.9	11.3	9.4	7.6	12.7	16.8	20.6	23.7	24.9	25.7
vie_Latn	0.1	0.1	0.1	0.0	0.0	0.0	0.3	0.3	0.1	0.1	0.1	0.0
zsm_Latn	2.1	1.9	1.5	1.2	0.7	0.4	6.1	7.8	8.9	9.9	10.4	9.2

Table 13: Gain in **compression** (%) of **continued tokenizer training** over **naive extension** from an independent tokenizer. Continues in Table 14.

	Mistral Nemo						Llama-2					
Lang.	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	1.6	1.8	1.7	1.4	1.2	1.0	3.0	4.1	5.0	6.0	6.6	6.7
als_Latn	3.3	2.9	2.9	3.0	2.3	1.6	8.0	9.3	10.3	11.5	10.4	10.3
arb_Arab	0.2	0.3	0.2	0.3	0.3	0.3	1.6	0.7	0.3	0.1	-0.2	-0.2
ary_Arab	0.4	-0.0	0.0	0.1	0.1	0.3	0.8	0.4	0.2	0.1	-0.1	-0.2
azj_Latn	3.6	4.5	5.2	5.5	5.1	4.3	11.8	13.8	18.0	20.4	22.4	22.1
bel_Cyrl	6.1	7.1	7.2	7.0	6.1	4.6	1.7	2.6	3.1	4.0	5.1	5.6
ben_Beng	0.7	0.6	0.6	0.6	0.7	0.7	0.3	0.1	-0.2	-0.4	-0.5	-0.8
bos_Latn	4.5	6.0	6.3	6.5	6.0	5.4	8.6	11.9	15.6	19.2	22.6	25.7
bul_Cyrl	2.2	2.5	2.5	2.5	2.2	1.9	1.8	2.2	2.7	3.3	3.5	4.4
cat_Latn	1.7	2.0	1.7	1.5	1.4	0.8	2.9	3.4	4.1	4.3	4.7	4.8
ces_Latn	2.6	3.8	4.5	4.8	5.2	4.4	5.1	6.9	9.0	11.5	13.4	14.8
cmn_Hani	0.1	0.1	-0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	-0.4	-0.5
dan_Latn	1.3	1.5	1.6	1.5	1.6	1.5	2.4	3.0	4.0	4.6	5.0	5.4
deu_Latn	0.6	0.9	1.1	1.1	1.3	1.1	1.6	2.1	2.9	3.6	4.4	5.1
ekk_Latn	3.3	3.5	4.0	4.7	4.7	4.5	5.9	7.2	8.3	9.6	10.7	12.4
ell_Grek	1.5	2.0	2.5	2.8	2.4	1.7	0.5	-0.2	-0.2	-0.2	-0.2	0.0
eus_Latn	4.2	5.1	5.7	6.1	5.4	4.5	12.3	15.7	19.0	21.1	23.1	24.8
fas_Arab	0.8	0.7	0.4	0.3	0.2	0.1	0.8	0.4	0.5	-0.0	0.0	0.1
fin_Latn	4.7	5.6	7.4	8.2	8.6	9.0	7.7	10.5	13.3	16.0	18.7	21.0
fra_Latn	0.7	0.8	0.9	0.7	0.7	0.5	1.9	2.3	2.6	3.0	3.3	3.6
glg_Latn	1.3	1.5	1.4	1.1	1.0	0.7	2.7	3.1	3.8	4.3	4.7	4.9
guj_Gujr	1.0	1.2	1.2	1.3	1.0	0.8	1.2	0.6	0.3	0.2	0.2	-0.0
heb_Hebr	9.6	13.6	16.2	17.1	15.6	13.2	0.1	0.0	0.0	-0.1	-0.1	-0.3
hin_Deva	0.4	0.4	0.5	0.3	0.3	0.2	2.6	1.2	0.6	0.0	0.0	0.0
hrv_Latn	4.6	5.7	6.9	7.8	7.2	6.0	9.3	13.0	17.3	21.5	26.1	29.9
hun_Latn	1.8	2.5	3.1	3.3	3.8	3.9	4.0	5.7	7.4	9.2	11.2	12.6
hye_Armn	0.3	0.3	0.2	0.3	0.3	0.2	0.0	0.2	-0.0	-0.1	-0.1	-0.1
ind_Latn	1.8	1.6	1.7	1.1	0.8	0.5	7.4	9.2	11.5	12.9	11.3	9.8
isl_Latn	2.6	2.8	3.2	3.2	3.6	3.6	3.2	4.2	4.9	5.4	6.8	7.5
ita_Latn	3.0	3.3	3.4	3.1	2.2	1.9	4.8	5.7	7.2	8.6	9.7	11.0
jpn_Jpan	1.4	1.4	1.5	1.5	1.6	1.3	0.0	0.0	0.0	-1.0	-2.2	-2.9
kan_Knda	0.2	-0.0	0.3	0.2	0.4	0.4	1.1	0.4	0.2	0.0	0.1	0.2
kat_Geor	1.2	1.1	1.0	1.0	0.7	0.6	0.9	0.4	0.3	-0.0	-0.3	-0.1
kaz_Cyrl	9.0	10.3	10.3	8.6	6.9	4.7	17.4	20.6	24.3	23.1	20.0	17.2
khk_Cyrl	6.5	7.6	7.3	6.0	4.3	3.1	13.1	15.5	17.4	18.8	19.0	18.1
khm_Khmr	0.0	0.1	0.2	0.2	0.2	0.2	0.0	4.8	1.6	0.7	0.3	0.2
kir_Cyrl	9.4	11.1	10.5	9.1	7.3	5.4	20.8	19.8	24.0	26.0	26.2	25.3
kor_Hang	0.0	0.0	0.1	0.1	0.1	0.1	0.0	0.0	0.0	0.1	-0.3	-0.2
lit_Latn	3.5	4.7	5.0	5.1	4.8	4.0	5.1	7.5	9.5	12.8	15.0	16.6
lvs_Latn	3.5	4.0	4.5	4.7	4.2	3.2	6.4	7.8	8.3	9.7	11.3	12.0
mal_Mlym	0.3	0.4	0.5	0.5	0.5	0.5	1.2	0.4	0.1	0.1	0.0	-0.0
mar_Deva	1.1	1.5	1.3	1.4	1.2	0.9	1.1	0.3	0.1	-0.1	-0.4	-0.7
mkd_Cyrl	2.0	2.2	2.4	2.7	2.4	1.9	2.1	2.5	2.6	3.1	3.8	4.2
mya_Mymr	8.2	11.0	14.6	17.6	21.6	23.7	5.9	2.3	1.0	-0.3	-0.7	-1.6
nld_Latn	1.3	1.7	1.7	1.5	1.3	1.0	3.3	4.0	5.1	6.6	7.1	7.0
nno_Latn	1.6	1.7	1.7	1.6	1.6	1.4	3.5	3.9	4.5	5.2	5.8	5.3
nob_Latn	0.9	1.2	1.1	1.3	1.3	1.3	2.2	2.7	3.5	4.2	4.5	5.0
npi_Deva	1.6	1.5	1.7	1.7	1.5	1.3	0.5	0.6	0.2	0.0	-0.1	-0.1
ory_Orya	0.0	0.0	0.0	0.0	0.0	0.0	1.2	0.6	0.3	0.1	-0.0	0.0
pan_Guru	0.1	0.0	0.1	-0.0	0.0	0.0	1.1	0.4	0.1	-0.0	-0.0	-0.1
pol_Latn	3.8	5.1	6.2	7.2	7.3	6.7	7.5	11.0	14.9	19.3	22.4	25.9
por_Latn	1.1	1.2	1.1	1.0	0.8	0.5	1.9	2.6	3.3	3.8	4.1	4.3
ron_Latn	3.5	4.1	4.2	4.3	3.7	2.5	6.8	7.7	9.1	10.7	11.7	11.7
rus_Cyrl	1.9	2.2	2.6	2.7	2.8	2.4	1.4	1.5	2.0	2.5	2.6	3.2
sin_Sinh	0.0	0.0	0.0	0.1	0.0	0.0	1.0	0.3	0.2	0.1	-0.0	0.0
slk_Latn	4.5	5.1	5.4	5.8	5.7	5.1	6.3	8.4	10.9	13.0	15.2	15.8
slv_Latn	4.5	5.7	6.4	6.6	6.2	5.3	8.3	10.9	13.9	17.1	19.3	21.3
spa_Latn	1.4	1.5	1.4	1.4	1.0	0.7	2.9	3.5	3.9	4.4	5.1	5.4
srp_Cyrl	5.5	6.6	7.9	8.0	8.0	6.6	1.6	1.8	2.3	2.8	3.6	4.0
swe_Latn	0.7	1.2	1.2	1.6	1.6	1.3	1.9	2.6	3.3	4.3	4.5	4.5
swh_Latn	8.1	9.1	9.3	9.6	8.4	6.7	20.6	27.0	34.4	41.5	43.7	44.4
tam_Taml	0.3	0.3	0.6	0.4	0.5	0.4	4.3	1.4	0.6	0.1	-0.1	-0.3
tel_Telu	1.1	1.4	1.9	1.6	1.5	1.2	1.6	0.5	0.5	0.2	-0.0	-0.3
tha_Thai	35.8	53.6	70.2	86.2	100.4	110.9	9.6	3.9	1.5	0.5	0.2	0.3
tur_Latn	2.1	2.9	3.4	3.5	3.6	3.0	8.9	10.4	12.4	14.4	15.8	15.2
ukr_Cyrl	4.0	4.8	5.1	5.3	5.2	4.5	1.8	2.1	2.5	3.3	3.8	4.0
urd_Arab	0.9	1.0	0.6	0.5	0.3	0.1	0.6	0.2	-0.1	-0.1	-0.1	-0.0
uzn_Latn	7.2	7.9	8.2	8.1	7.5	6.3	13.6	17.1	21.1	22.9	23.6	21.8
vie_Latn	0.1	0.1	0.0	0.0	0.0	0.0	9.3	4.1	2.5	1.8	0.9	0.2
zsm_Latn	2.2	2.0	1.4	1.0	0.5	0.4	9.0	10.2	11.8	11.3	11.5	9.0

Table 14: Gain in **compression** (%) of **continued tokenizer training** over **naive extension** from an independent tokenizer.

	Llama-3 extension						Qwen-2.5 extension					
Lang.	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	-74	-156	-357	-875	-1858	-3692	-72	-155	-350	-882	-1859	-3676
als_Latn	-95	-229	-478	-983	-2004	-4071	-94	-231	-491	-1007	-2045	-4127
arb_Arab	-23	-58	-141	-293	-547	-1039	-15	-52	-162	-366	-733	-1442
ary_Arab	-27	-52	-120	-264	-510	-954	-28	-63	-163	-358	-728	-1411
azj_Latn	-161	-359	-851	-1714	-3746	-7746	-137	-309	-751	-1591	-3525	-7402
bel_Cyrl	-54	-132	-313	-699	-1509	-3216	-22	-89	-241	-596	-1330	-2772
ben_Beng	0	0	0	0	-46	-225	0	0	0	-1	-53	-231
bos_Latn	-62	-174	-520	-1449	-3599	-8484	-67	-184	-567	-1534	-3737	-8707
bul_Cyrl	-89	-198	-516	-1312	-3364	-7808	-111	-274	-704	-1751	-4177	-8923
cat_Latn	-60	-133	-342	-752	-1635	-3432	-60	-132	-337	-748	-1623	-3404
ces_Latn	-118	-183	-377	-890	-2052	-4599	-64	-166	-392	-951	-2273	-5015
cmn_Hani	-2	-3	-5	-15	-25	-74	-16	-36	-66	-122	-211	-369
dan_Latn	-55	-141	-326	-747	-1578	-3522	-53	-140	-323	-752	-1585	-3516
deu_Latn	-15	-67	-239	-659	-1511	-3362	-16	-70	-247	-666	-1524	-3375
ekk_Latn	-78	-175	-359	-899	-2084	-4612	-77	-177	-361	-897	-2092	-4628
ell_Grek	-23	-59	-148	-324	-680	-1230	0	-4	-13	-31	-73	-19
eus_Latn	-115	-302	-682	-1604	-3492	-7483	-111	-299	-689	-1631	-3533	-7564
fas_Arab	-27	-64	-138	-286	-562	-1070	-38	-116	-261	-582	-1121	-2116
fin_Latn	-57	-184	-439	-1097	-2525	-5813	-58	-186	-443	-1105	-2536	-5844
fra_Latn	-17	-57	-178	-489	-1153	-2505	-17	-57	-171	-476	-1142	-2488
glg_Latn	-36	-100	-216	-501	-1147	-2670	-36	-105	-227	-497	-1147	-2663
guj_Gujr	0	0	-4	-14	-73	-273	0	0	-4	-14	-64	-271
heb_Hebr	-133	-310	-672	-1480	-3254	-5762	-36	-132	-482	-1413	-3795	-8746
hin_Deva	0	0	-2	-40	-239	-655	0	0	-2	-25	-218	-632
hrv_Latn	-65	-224	-656	-1691	-4147	-9578	-71	-235	-694	-1780	-4277	-9769
hun_Latn	-75	-174	-411	-1043	-2287	-5145	-73	-167	-404	-1018	-2266	-5054
hye_Armn	0	0	-1	-1	-2	-22	-5	-10	-21	-68	-143	-259
ind_Latn	-60	-148	-361	-782	-1629	-3248	-59	-144	-356	-768	-1608	-3232
isl_Latn	-86	-160	-427	-923	-1983	-4369	-83	-157	-418	-911	-1960	-4286
ita_Latn	-41	-145	-376	-1051	-2596	-6332	-39	-144	-374	-1046	-2600	-6314
jpn_Jpan	-14	-23	-37	-89	-232	-479	-4	-10	-36	-86	-223	-552
kan_Knda	0	0	-5	-13	-57	-275	0	0	-5	-13	-58	-281
kat_Geor	0	-8	-12	-17	-37	-104	-5	-32	-75	-176	-414	-828
kaz_Cyrl	-74	-198	-409	-902	-1971	-3939	-61	-182	-367	-855	-1810	-3790
khk_Cyrl	-68	-152	-389	-914	-1833	-3621	-67	-145	-358	-806	-1644	-3116
khm_Khmr	0	0	0	-3	-18	-119	0	0	0	-2	-23	-136
kir_Cyrl	-68	-155	-402	-938	-2005	-4238	-60	-135	-361	-825	-1774	-3810
kor_Hang	-13	-21	-36	-77	-146	-307	-5	-15	-32	-71	-142	-304
lit_Latn	-71	-177	-455	-1118	-2456	-5307	-64	-178	-450	-1105	-2447	-5255
lvs_Latn	-77	-188	-338	-786	-1817	-3869	-66	-177	-328	-783	-1796	-3800
mal_Mlym	0	-2	-5	-13	-74	-387	0	-2	-5	-15	-71	-384
mar_Deva	-3	-14	-17	-41	-163	-521	-2	-1	-5	-18	-130	-501
mkd_Cyrl	-91	-263	-627	-1562	-3647	-8282	-103	-319	-767	-1830	-4215	-9192
mya_Mymr	-16	-20	-43	-85	-200	-518	-15	-21	-46	-90	-213	-549
nld_Latn	-67	-168	-388	-866	-1757	-3742	-65	-163	-384	-863	-1751	-3736
nno_Latn	-46	-108	-253	-552	-1303	-3095	-48	-109	-249	-564	-1317	-3072
nob_Latn	-36	-101	-255	-530	-1208	-2639	-38	-101	-255	-530	-1211	-2626
npi_Deva	-1	-3	-6	-18	-49	-169	0	0	0	-3	-20	-131
ory_Orya	-2	-3	-3	-2	-46	-296	-2	-3	-3	-4	-74	-324
pan_Guru	0	-1	-4	-18	-91	-340	0	-1	-4	-18	-89	-354
pol_Latn	-56	-192	-567	-1408	-3443	-7898	-56	-195	-545	-1363	-3363	-7806
por_Latn	-19	-52	-158	-419	-998	-2165	-21	-54	-166	-428	-1003	-2162
ron_Latn	-50	-138	-337	-817	-1975	-4487	-48	-140	-338	-815	-1981	-4499
rus_Cyrl	-81	-134	-244	-462	-922	-1898	-22	-59	-139	-328	-791	-1699
sin_Sinh	-58	-110	-206	-409	-783	-1494	-60	-110	-207	-410	-783	-1494
slk_Latn	-103	-197	-443	-1019	-2367	-5391	-52	-152	-408	-990	-2395	-5512
slv_Latn	-91	-229	-599	-1491	-3524	-7898	-73	-210	-595	-1479	-3545	-7863
spa_Latn	-17	-44	-170	-475	-1224	-3033	-18	-44	-165	-473	-1218	-3030
srp_Cyrl	-45	-119	-284	-625	-1525	-3351	-23	-90	-267	-595	-1478	-3248
swe_Latn	-68	-156	-350	-751	-1459	-3064	-68	-154	-341	-737	-1437	-3052
swh_Latn	-157	-443	-1146	-2513	-5583	-12025	-163	-468	-1185	-2581	-5670	-12159
tam_Taml	0	0	-3	-34	-156	-570	0	0	-3	-37	-151	-607
tel_Telu	0	0	-2	-13	-81	-407	0	0	-2	-13	-76	-405
tha_Thai	-9	-15	-28	-54	-105	-250	-2	-7	-34	-70	-136	-334
tur_Latn	-232	-359	-645	-1245	-2588	-5275	-51	-141	-358	-856	-1974	-4377
ukr_Cyrl	-76	-123	-225	-473	-985	-2063	-28	-75	-192	-470	-1008	-2206
urd_Arab	-48	-98	-192	-420	-783	-1393	-53	-121	-310	-596	-1012	-1664
uzn_Latn	-105	-297	-733	-1648	-3510	-7571	-105	-288	-736	-1677	-3577	-7690
vie_Latn	-35	-48	-65	-148	-375	-1010	-6	-10	-26	-121	-343	-982
zsm_Latn	-66	-145	-369	-823	-1675	-3153	-65	-141	-360	-802	-1641	-3119

Table 15: **Change in unreachable tokens through merges** of **continued tokenizer training** compared to **naive extension** from an independent tokenizer. Continues in Table 16.

	Mistral Nemo extension						Llama-2 extension					
Lang.	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	-62	-132	-329	-793	-1634	-3290	-59	-149	-330	-753	-1617	-3187
als_Latn	-83	-197	-437	-880	-1833	-3661	-78	-226	-514	-1070	-2074	-4218
arb_Arab	-9	-28	-58	-114	-245	-577	0	0	0	-2	-4	-12
ary_Arab	-10	-24	-55	-103	-264	-550	0	0	0	-1	-1	-6
azj_Latn	-104	-225	-552	-1167	-2577	-5327	-128	-255	-631	-1474	-3261	-6616
bel_Cyrl	-52	-158	-408	-950	-2178	-4767	-22	-65	-138	-319	-779	-1708
ben_Beng	-6	-19	-45	-93	-167	-358	0	0	0	0	0	-1
bos_Latn	-24	-79	-292	-908	-2462	-6251	-56	-197	-545	-1524	-3658	-8530
bul_Cyrl	-21	-60	-161	-426	-1081	-2623	-16	-47	-125	-300	-673	-1510
cat_Latn	-33	-84	-204	-480	-1058	-2439	-37	-115	-285	-648	-1442	-3082
ces_Latn	-31	-85	-254	-745	-1809	-4197	-52	-150	-377	-920	-2199	-4876
cmn_Hani	0	0	-2	-9	-32	-80	0	0	0	0	0	-7
dan_Latn	-53	-123	-285	-633	-1360	-2964	-56	-132	-304	-694	-1424	-3007
deu_Latn	-15	-43	-136	-376	-922	-2237	-21	-73	-209	-532	-1155	-2543
ekk_Latn	-73	-160	-367	-863	-1958	-4310	-66	-176	-382	-911	-2044	-4528
ell_Grek	-23	-58	-177	-424	-1023	-2265	0	-1	-1	-6	-8	-37
eus_Latn	-76	-206	-518	-1244	-2835	-6223	-113	-320	-724	-1622	-3586	-7406
fas_Arab	-16	-51	-144	-348	-725	-1440	0	0	0	-1	-4	-30
fin_Latn	-33	-104	-287	-811	-2016	-4868	-63	-164	-422	-1014	-2390	-5509
fra_Latn	-19	-46	-107	-255	-613	-1578	-28	-78	-185	-445	-1023	-2203
glg_Latn	-29	-70	-156	-335	-784	-1888	-41	-102	-232	-551	-1179	-2659
guj_Gujr	-20	-43	-80	-165	-301	-616	0	0	-2	-3	-10	-25
heb_Hebr	-28	-115	-453	-1511	-4173	-9750	0	0	0	0	0	-1
hin_Deva	-17	-27	-55	-104	-202	-369	0	0	0	0	-8	-62
hrv_Latn	-43	-139	-431	-1209	-3044	-7548	-74	-243	-686	-1714	-4212	-9613
hun_Latn	-40	-102	-292	-675	-1548	-3624	-59	-154	-333	-847	-1906	-4161
hye_Armn	-3	-10	-28	-54	-92	-201	0	0	-1	-1	-3	-11
ind_Latn	-38	-105	-256	-568	-1339	-2807	-70	-159	-398	-776	-1574	-3100
isl_Latn	-74	-158	-371	-799	-1760	-3956	-66	-151	-336	-736	-1665	-3615
ita_Latn	-18	-56	-180	-589	-1579	-4322	-30	-122	-349	-919	-2275	-5737
jpn_Jpan	-7	-12	-24	-59	-160	-369	0	0	0	0	-1	-5
kan_Knda	-5	-9	-20	-31	-57	-129	0	0	0	-3	-4	-15
kat_Geor	-21	-43	-70	-152	-241	-404	0	0	0	-1	-1	-15
kaz_Cyrl	-72	-213	-496	-1056	-2328	-4474	-100	-251	-589	-1258	-2536	-4867
khk_Cyrl	-72	-223	-548	-1180	-2537	-4966	-69	-213	-554	-1247	-2642	-5135
khm_Khmr	-5	-7	-12	-21	-34	-65	0	0	0	-1	-8	-32
kir_Cyrl	-100	-232	-550	-1317	-2713	-5455	-109	-248	-632	-1463	-3069	-6264
kor_Hang	-2	-5	-8	-13	-28	-66	0	0	0	0	-1	-10
lit_Latn	-55	-143	-399	-959	-2146	-4796	-51	-156	-426	-1069	-2390	-5220
lvs_Latn	-71	-169	-338	-793	-1788	-3718	-43	-159	-330	-775	-1788	-3692
mal_Mlym	-1	-4	-14	-36	-67	-143	0	0	-4	-5	-11	-16
mar_Deva	-17	-50	-116	-245	-449	-926	0	-2	-3	-6	-12	-29
mkd_Cyrl	-25	-72	-208	-476	-1187	-2746	-22	-71	-165	-358	-829	-1754
mya_Mymr	-71	-163	-435	-1127	-2617	-5592	0	0	-1	-4	-11	-29
nld_Latn	-38	-106	-289	-677	-1440	-3137	-59	-142	-356	-804	-1584	-3169
nno_Latn	-41	-91	-229	-476	-1130	-2711	-43	-106	-257	-571	-1238	-2679
nob_Latn	-35	-81	-203	-432	-968	-2148	-35	-108	-249	-556	-1116	-2302
npi_Deva	-30	-53	-110	-197	-380	-800	0	0	0	0	0	-2
ory_Orya	0	0	0	0	-1	-8	0	0	0	0	-2	-7
pan_Guru	-1	-6	-5	-8	-23	-40	0	0	-1	-2	-11	-33
pol_Latn	-39	-127	-324	-856	-2388	-6054	-66	-198	-576	-1400	-3410	-7708
por_Latn	-13	-38	-126	-292	-663	-1594	-25	-78	-206	-518	-1149	-2424
ron_Latn	-36	-101	-258	-672	-1580	-3697	-41	-127	-355	-874	-2133	-4756
rus_Cyrl	-19	-56	-187	-467	-1155	-2795	-17	-45	-101	-270	-573	-1281
sin_Sinh	0	0	-1	-1	-2	-4	0	0	0	-2	-3	-10
slk_Latn	-43	-130	-313	-786	-1864	-4326	-45	-134	-380	-955	-2237	-4932
slv_Latn	-44	-125	-373	-1019	-2612	-6297	-63	-198	-538	-1443	-3468	-7712
spa_Latn	-6	-18	-69	-207	-628	-1834	-24	-73	-213	-524	-1276	-3009
srp_Cyrl	-37	-127	-406	-1146	-2927	-6961	-18	-58	-122	-300	-765	-1805
swe_Latn	-71	-135	-291	-566	-1137	-2516	-64	-135	-287	-589	-1127	-2282
swh_Latn	-125	-356	-983	-2228	-5034	-10967	-160	-458	-1199	-2712	-5683	-11956
tam_Taml	-2	-11	-28	-69	-151	-237	0	0	0	0	-1	-7
tel_Telu	-18	-36	-78	-181	-335	-631	0	0	0	-2	-2	-17
tha_Thai	-250	-567	-1319	-2768	-5888	-12112	0	0	-1	-3	-14	-38
tur_Latn	-49	-115	-285	-673	-1662	-3679	-91	-201	-438	-953	-2229	-4705
ukr_Cyrl	-31	-92	-302	-793	-1862	-4242	-22	-49	-113	-299	-620	-1324
urd_Arab	-31	-78	-177	-392	-781	-1392	0	0	0	-1	-4	-37
uzn_Latn	-64	-214	-556	-1316	-2911	-6400	-128	-302	-766	-1647	-3490	-6834
vie_Latn	-5	-7	-16	-98	-281	-851	-13	-32	-59	-115	-333	-991
zsm_Latn	-49	-118	-302	-658	-1412	-2821	-48	-136	-342	-767	-1602	-3063

Table 16: **Change in unreachable tokens through merges** of **continued tokenizer training** compared to **naive extension** from an independent tokenizer.

	Llama-3 extension							Qwen-2.5 extension						
Lang.	**0**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**0**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	3.093	3.630	3.828	4.062	4.301	4.529	4.720	3.051	3.579	3.773	4.004	4.239	4.463	4.649
als_Latn	2.625	3.550	3.770	4.036	4.328	4.645	4.936	2.579	3.484	3.710	3.976	4.266	4.577	4.866
arb_Arab	4.705	5.138	5.438	5.867	6.385	6.941	7.542	4.697	5.158	5.431	5.838	6.313	6.855	7.438
ary_Arab	4.450	4.813	5.034	5.349	5.775	6.269	6.797	4.405	4.803	5.027	5.316	5.711	6.193	6.704
azj_Latn	2.537	3.787	4.120	4.516	4.936	5.386	5.835	2.420	3.624	3.966	4.388	4.834	5.281	5.730
bel_Cyrl	3.866	5.204	5.640	6.184	6.845	7.547	8.265	3.301	4.875	5.368	5.962	6.658	7.402	8.140
ben_Beng	2.203	5.023	5.196	5.291	5.345	5.366	5.375	2.483	5.159	5.336	5.429	5.482	5.503	5.513
bos_Latn	2.780	3.265	3.476	3.714	3.998	4.330	4.663	2.651	3.161	3.377	3.635	3.920	4.255	4.588
bul_Cyrl	4.527	5.746	6.087	6.511	7.054	7.675	8.286	4.115	5.395	5.783	6.258	6.856	7.511	8.139
cat_Latn	3.219	3.705	3.865	4.066	4.291	4.530	4.721	3.194	3.662	3.816	4.017	4.236	4.473	4.657
ces_Latn	3.485	3.700	3.827	4.013	4.286	4.631	5.020	2.496	3.193	3.449	3.743	4.081	4.483	4.897
cmn_Hani	3.471	3.651	3.788	3.980	4.250	4.559	4.903	4.361	4.493	4.548	4.614	4.706	4.843	5.032
dan_Latn	3.157	3.579	3.752	3.986	4.230	4.483	4.729	3.112	3.527	3.696	3.925	4.163	4.414	4.654
deu_Latn	3.638	3.877	4.047	4.259	4.529	4.828	5.115	3.631	3.855	4.012	4.215	4.476	4.763	5.042
ekk_Latn	2.644	3.282	3.509	3.767	4.084	4.441	4.798	2.607	3.230	3.454	3.711	4.023	4.370	4.717
ell_Grek	4.736	5.443	5.841	6.433	7.148	7.924	8.683	2.123	4.647	5.317	6.054	6.895	7.748	8.538
eus_Latn	2.774	3.581	3.851	4.174	4.516	4.890	5.247	2.722	3.519	3.787	4.106	4.443	4.810	5.158
fas_Arab	5.404	5.769	6.030	6.377	6.755	7.063	7.284	3.140	4.705	5.191	5.753	6.297	6.683	6.914
fin_Latn	2.704	3.405	3.661	3.957	4.323	4.732	5.158	2.674	3.359	3.611	3.901	4.258	4.657	5.072
fra_Latn	3.771	4.025	4.172	4.363	4.596	4.820	5.007	3.756	3.996	4.134	4.313	4.534	4.749	4.929
glg_Latn	3.536	3.924	4.084	4.310	4.558	4.818	5.042	3.513	3.894	4.042	4.262	4.497	4.750	4.967
guj_Gujr	1.590	5.019	5.226	5.340	5.401	5.429	5.439	1.779	4.994	5.193	5.302	5.361	5.390	5.400
heb_Hebr	1.849	4.150	4.655	5.220	5.818	6.442	7.079	4.497	4.949	5.197	5.540	5.959	6.456	7.007
hin_Deva	4.920	5.089	5.139	5.175	5.195	5.206	5.209	2.764	4.902	5.064	5.130	5.161	5.173	5.177
hrv_Latn	2.757	3.263	3.462	3.723	4.022	4.360	4.720	2.629	3.157	3.372	3.638	3.944	4.280	4.639
hun_Latn	2.655	3.302	3.581	3.913	4.304	4.709	5.135	2.605	3.247	3.524	3.852	4.239	4.638	5.051
hye_Armn	1.147	4.950	5.753	6.603	7.472	8.315	9.146	1.851	5.040	5.765	6.580	7.412	8.233	9.019
ind_Latn	3.410	4.049	4.338	4.714	5.080	5.402	5.643	3.350	3.974	4.258	4.623	4.987	5.305	5.540
isl_Latn	2.523	3.264	3.514	3.795	4.107	4.457	4.800	2.477	3.215	3.462	3.735	4.049	4.395	4.729
ita_Latn	3.533	3.910	4.094	4.337	4.613	4.887	5.116	3.502	3.868	4.047	4.283	4.552	4.821	5.043
jpn_Jpan	4.071	4.426	4.639	4.950	5.361	5.866	6.476	4.313	4.687	4.874	5.144	5.480	5.910	6.447
kan_Knda	1.560	5.260	5.488	5.615	5.709	5.734	5.743	1.964	5.231	5.453	5.576	5.667	5.691	5.706
kat_Geor	1.470	6.859	7.975	9.356	10.827	12.350	13.899	2.718	6.926	7.989	9.267	10.693	12.215	13.708
kaz_Cyrl	2.983	5.550	6.254	7.026	7.825	8.674	9.492	3.007	5.371	6.059	6.841	7.664	8.504	9.314
khk_Cyrl	2.869	5.445	6.159	7.007	7.870	8.690	9.423	2.962	5.291	5.996	6.859	7.723	8.541	9.268
khm_Khmr	1.837	5.003	5.323	5.541	5.676	5.752	5.781	2.435	5.001	5.305	5.517	5.649	5.720	5.748
kir_Cyrl	3.175	5.554	6.183	6.950	7.727	8.552	9.344	3.292	5.344	5.976	6.764	7.563	8.393	9.174
kor_Hang	3.902	4.207	4.413	4.699	5.070	5.487	5.936	3.505	4.028	4.282	4.606	4.982	5.395	5.838
lit_Latn	2.389	3.299	3.557	3.865	4.236	4.657	5.088	2.469	3.233	3.496	3.805	4.172	4.584	5.008
lvs_Latn	2.341	3.283	3.562	3.896	4.272	4.688	5.094	2.331	3.220	3.500	3.829	4.201	4.609	5.008
mal_Mlym	1.696	5.394	5.662	5.825	5.930	5.969	5.984	2.063	5.365	5.623	5.777	5.894	5.931	5.945
mar_Deva	4.677	5.265	5.348	5.414	5.461	5.482	5.491	2.789	5.067	5.260	5.360	5.416	5.441	5.453
mkd_Cyrl	4.349	5.733	6.113	6.591	7.195	7.851	8.481	4.005	5.428	5.837	6.371	7.012	7.686	8.333
mya_Mymr	1.474	5.304	5.497	5.591	5.635	5.651	5.661	1.888	5.309	5.486	5.571	5.611	5.627	5.635
nld_Latn	3.402	3.835	4.038	4.295	4.574	4.828	5.044	3.364	3.784	3.983	4.236	4.508	4.758	4.969
nno_Latn	3.079	3.595	3.768	3.977	4.221	4.465	4.688	3.031	3.538	3.712	3.916	4.156	4.398	4.615
nob_Latn	3.196	3.600	3.763	3.980	4.222	4.460	4.689	3.149	3.545	3.705	3.915	4.153	4.389	4.613
npi_Deva	4.837	5.385	5.483	5.572	5.626	5.653	5.668	2.784	5.175	5.371	5.503	5.574	5.604	5.626
ory_Orya	1.094	5.192	5.401	5.506	5.567	5.591	5.614	1.349	5.182	5.392	5.497	5.554	5.579	5.601
pan_Guru	1.621	4.748	4.922	5.003	5.042	5.058	5.064	1.712	4.796	4.970	5.055	5.096	5.112	5.119
pol_Latn	2.886	3.342	3.579	3.886	4.236	4.628	5.057	3.041	3.412	3.623	3.889	4.213	4.582	4.997
por_Latn	3.686	3.951	4.105	4.308	4.547	4.804	5.017	3.682	3.926	4.073	4.260	4.495	4.741	4.947
ron_Latn	3.092	3.728	3.918	4.166	4.435	4.727	4.996	3.056	3.679	3.867	4.113	4.377	4.662	4.924
rus_Cyrl	5.928	6.255	6.435	6.720	7.164	7.750	8.413	5.371	5.825	6.089	6.433	6.946	7.561	8.253
sin_Sinh	1.476	5.029	5.268	5.422	5.494	5.524	5.540	1.822	4.907	5.131	5.273	5.340	5.369	5.383
slk_Latn	2.871	3.470	3.667	3.912	4.215	4.571	4.961	2.510	3.184	3.443	3.751	4.076	4.470	4.859
slv_Latn	2.744	3.259	3.447	3.679	3.960	4.262	4.592	2.599	3.147	3.353	3.594	3.884	4.185	4.514
spa_Latn	3.821	4.055	4.195	4.376	4.598	4.824	5.008	3.806	4.030	4.161	4.329	4.545	4.766	4.942
srp_Cyrl	4.013	5.356	5.717	6.151	6.668	7.278	7.919	3.694	5.050	5.455	5.925	6.492	7.130	7.781
swe_Latn	3.223	3.607	3.785	4.023	4.299	4.609	4.883	3.177	3.556	3.727	3.962	4.231	4.538	4.804
swh_Latn	2.636	3.495	3.797	4.130	4.480	4.771	4.997	2.586	3.442	3.734	4.067	4.409	4.695	4.917
tam_Taml	2.029	5.590	5.723	5.785	5.806	5.811	5.815	2.497	5.565	5.693	5.751	5.771	5.776	5.781
tel_Telu	1.570	5.017	5.163	5.238	5.288	5.303	5.309	1.831	4.989	5.133	5.205	5.253	5.268	5.274
tha_Thai	6.146	6.726	7.068	7.543	8.048	8.551	9.042	5.197	6.494	6.913	7.414	7.948	8.460	8.953
tur_Latn	3.882	4.194	4.332	4.551	4.848	5.189	5.554	3.298	3.809	4.041	4.338	4.683	5.054	5.437
ukr_Cyrl	5.512	5.910	6.137	6.487	6.955	7.554	8.206	3.564	5.024	5.445	5.965	6.594	7.282	7.998
urd_Arab	2.774	5.256	5.621	6.048	6.463	6.813	7.065	2.664	4.779	5.251	5.807	6.294	6.691	6.963
uzn_Latn	2.595	3.424	3.719	4.049	4.417	4.803	5.132	2.522	3.346	3.644	3.973	4.345	4.725	5.049
vie_Latn	4.856	5.076	5.175	5.244	5.283	5.302	5.313	4.631	5.008	5.110	5.177	5.215	5.234	5.245
zsm_Latn	3.357	4.163	4.514	4.906	5.311	5.627	5.815	3.301	4.091	4.431	4.815	5.214	5.523	5.710

Table 17: Compression (*bytes per token*) on FLORES for tokenizer extension with continued BPE training. The columns represent different number of added tokens where 0 is the original tokenizer. Continues is Table 18.

	Mistral Nemo							Llama-2						
Lang.	**0**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**0**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**
afr_Latn	3.343	3.728	3.886	4.087	4.294	4.499	4.666	2.932	3.461	3.660	3.903	4.145	4.379	4.580
als_Latn	3.049	3.613	3.817	4.061	4.332	4.623	4.887	2.259	3.384	3.616	3.887	4.188	4.503	4.797
arb_Arab	6.115	6.232	6.333	6.501	6.786	7.182	7.643	1.988	4.248	4.780	5.432	6.144	6.845	7.529
ary_Arab	5.411	5.600	5.704	5.871	6.104	6.428	6.857	1.982	3.996	4.479	4.958	5.549	6.139	6.702
azj_Latn	3.311	3.845	4.133	4.511	4.916	5.334	5.765	2.057	3.387	3.776	4.251	4.742	5.210	5.658
bel_Cyrl	4.522	5.344	5.744	6.241	6.850	7.530	8.214	3.685	5.032	5.498	6.068	6.746	7.467	8.154
ben_Beng	6.038	6.980	7.599	8.421	9.345	10.226	10.947	2.070	6.151	7.011	7.987	8.977	9.867	10.530
bos_Latn	3.210	3.462	3.608	3.801	4.036	4.329	4.633	2.597	3.099	3.318	3.578	3.867	4.199	4.541
bul_Cyrl	5.482	6.032	6.283	6.636	7.090	7.639	8.193	4.536	5.473	5.838	6.296	6.883	7.520	8.139
cat_Latn	3.716	3.926	4.039	4.189	4.353	4.542	4.689	3.175	3.505	3.653	3.848	4.047	4.269	4.442
ces_Latn	3.311	3.565	3.730	3.951	4.233	4.589	4.962	2.710	3.184	3.418	3.709	4.047	4.442	4.850
cmn_Hani	3.094	3.448	3.631	3.849	4.147	4.465	4.811	1.963	2.635	2.767	2.822	2.829	3.935	4.412
dan_Latn	3.380	3.673	3.818	4.007	4.216	4.453	4.679	2.900	3.379	3.570	3.798	4.045	4.307	4.558
deu_Latn	4.308	4.379	4.445	4.555	4.694	4.896	5.106	3.563	3.787	3.930	4.133	4.386	4.669	4.951
ekk_Latn	2.993	3.384	3.562	3.786	4.072	4.401	4.738	2.435	3.118	3.356	3.622	3.940	4.296	4.645
ell_Grek	4.925	5.478	5.840	6.390	7.075	7.825	8.557	1.853	4.613	5.287	6.028	6.876	7.722	8.509
eus_Latn	3.299	3.749	3.972	4.237	4.540	4.877	5.204	2.520	3.396	3.669	3.995	4.330	4.688	5.024
fas_Arab	5.585	5.915	6.136	6.445	6.767	7.050	7.257	1.812	4.406	4.997	5.689	6.281	6.712	6.948
fin_Latn	3.177	3.563	3.762	4.026	4.340	4.710	5.113	2.457	3.238	3.517	3.827	4.191	4.591	5.018
fra_Latn	4.467	4.525	4.567	4.634	4.727	4.856	4.978	3.578	3.787	3.918	4.089	4.290	4.489	4.676
glg_Latn	4.069	4.302	4.405	4.530	4.677	4.847	5.013	3.258	3.692	3.871	4.106	4.376	4.645	4.884
guj_Gujr	4.369	6.388	7.225	8.217	9.202	10.128	11.011	1.067	6.214	7.190	8.291	9.311	10.261	11.101
heb_Hebr	4.032	4.554	4.886	5.315	5.820	6.387	6.982	1.790	4.135	4.611	5.161	5.744	6.354	6.958
hin_Deva	6.743	7.418	7.889	8.540	9.221	9.878	10.436	2.365	6.202	7.142	8.135	9.047	9.783	10.330
hrv_Latn	3.187	3.444	3.592	3.804	4.068	4.363	4.683	2.571	3.097	3.307	3.576	3.887	4.225	4.584
hun_Latn	3.404	3.643	3.817	4.072	4.381	4.725	5.107	2.741	3.239	3.503	3.822	4.190	4.570	4.970
hye_Armn	5.063	5.755	6.210	6.836	7.543	8.288	9.041	1.703	4.831	5.546	6.285	7.037	7.756	8.418
ind_Latn	3.890	4.266	4.478	4.775	5.066	5.346	5.554	2.610	3.713	4.054	4.441	4.847	5.186	5.427
isl_Latn	2.737	3.313	3.532	3.793	4.095	4.423	4.751	2.341	3.135	3.396	3.675	3.997	4.333	4.683
ita_Latn	4.053	4.229	4.338	4.498	4.699	4.891	5.077	3.448	3.760	3.926	4.149	4.407	4.672	4.880
jpn_Jpan	3.827	4.299	4.544	4.881	5.292	5.774	6.361	2.408	2.853	2.895	2.899	4.394	5.143	5.744
kan_Knda	6.111	7.577	8.410	9.430	10.643	11.914	13.176	1.115	6.886	8.013	9.236	10.601	11.953	13.228
kat_Geor	5.853	7.439	8.352	9.488	10.828	12.247	13.708	2.584	6.908	8.015	9.365	10.817	12.330	13.815
kaz_Cyrl	4.384	5.869	6.518	7.238	7.965	8.701	9.408	3.199	5.504	6.201	7.003	7.797	8.585	9.329
khk_Cyrl	3.824	5.567	6.235	7.033	7.885	8.623	9.288	2.946	5.343	6.076	6.908	7.781	8.569	9.272
khm_Khmr	1.064	6.371	7.427	8.541	9.511	10.412	11.122	1.377	2.906	6.620	8.423	9.681	10.629	11.312
kir_Cyrl	4.127	5.787	6.404	7.101	7.804	8.550	9.271	3.285	5.468	6.120	6.878	7.667	8.437	9.201
kor_Hang	4.438	4.581	4.695	4.875	5.132	5.489	5.871	1.605	2.409	2.416	2.417	3.921	5.003	5.633
lit_Latn	2.894	3.390	3.620	3.916	4.252	4.635	5.037	2.386	3.175	3.439	3.762	4.138	4.551	4.972
lvs_Latn	2.756	3.369	3.614	3.932	4.284	4.665	5.043	2.258	3.156	3.442	3.773	4.165	4.577	4.979
mal_Mlym	5.774	7.625	8.602	9.802	11.106	12.557	13.990	2.391	7.199	8.407	9.727	11.179	12.705	14.161
mar_Deva	5.962	7.660	8.358	9.262	10.332	11.361	12.295	2.496	6.623	7.666	8.895	10.171	11.359	12.342
mkd_Cyrl	5.499	6.032	6.315	6.719	7.221	7.806	8.383	4.401	5.533	5.926	6.428	7.040	7.712	8.349
mya_Mymr	5.203	6.983	7.967	9.122	10.427	11.845	13.345	1.791	7.665	9.086	10.653	12.385	14.195	15.785
nld_Latn	3.764	4.014	4.170	4.360	4.585	4.806	4.991	3.243	3.670	3.864	4.108	4.377	4.639	4.865
nno_Latn	3.331	3.702	3.843	4.011	4.218	4.434	4.639	2.843	3.444	3.623	3.827	4.065	4.314	4.547
nob_Latn	3.431	3.689	3.828	4.005	4.217	4.435	4.639	2.902	3.408	3.585	3.803	4.049	4.297	4.542
npi_Deva	5.730	7.521	8.172	9.130	10.121	11.194	12.158	2.433	6.545	7.659	8.890	10.141	11.273	12.271
ory_Orya	1.010	6.522	7.674	8.922	10.176	11.318	12.272	1.006	6.715	7.874	9.125	10.376	11.422	12.290
pan_Guru	4.168	6.319	7.047	7.832	8.539	9.131	9.562	1.172	6.070	6.831	7.623	8.312	8.831	9.198
pol_Latn	3.382	3.644	3.807	4.024	4.311	4.651	5.038	2.809	3.256	3.497	3.789	4.141	4.519	4.933
por_Latn	4.169	4.292	4.375	4.491	4.643	4.816	4.973	3.349	3.697	3.869	4.091	4.340	4.608	4.851
ron_Latn	3.510	3.854	4.015	4.217	4.460	4.721	4.956	2.972	3.554	3.746	3.981	4.252	4.534	4.801
rus_Cyrl	6.220	6.443	6.606	6.859	7.224	7.729	8.317	5.167	5.671	5.966	6.382	6.886	7.515	8.204
sin_Sinh	1.010	6.048	6.922	7.945	8.930	9.770	10.491	1.431	6.084	6.973	8.016	9.016	9.864	10.565
slk_Latn	3.093	3.506	3.683	3.917	4.209	4.552	4.917	2.548	3.166	3.413	3.719	4.045	4.429	4.811
slv_Latn	3.088	3.385	3.539	3.738	3.980	4.255	4.555	2.595	3.107	3.314	3.566	3.844	4.143	4.472
spa_Latn	4.367	4.450	4.505	4.592	4.710	4.848	4.979	3.548	3.812	3.966	4.170	4.411	4.662	4.871
srp_Cyrl	5.215	5.689	5.949	6.307	6.739	7.295	7.878	4.440	5.233	5.600	6.037	6.589	7.213	7.860
swe_Latn	3.511	3.727	3.860	4.059	4.308	4.592	4.830	3.068	3.439	3.616	3.850	4.127	4.432	4.713
swh_Latn	2.736	3.514	3.794	4.114	4.442	4.717	4.928	2.398	3.315	3.626	3.976	4.337	4.631	4.860
tam_Taml	6.956	8.778	9.735	11.009	12.314	13.703	14.998	2.311	7.708	9.136	10.730	12.253	13.776	15.089
tel_Telu	5.623	7.353	8.180	9.160	10.292	11.538	12.654	1.066	6.712	7.757	8.985	10.343	11.651	12.769
tha_Thai	5.545	7.767	8.958	10.228	11.686	13.105	14.551	2.696	6.347	8.000	9.581	11.036	12.387	13.672
tur_Latn	3.564	3.899	4.119	4.408	4.741	5.111	5.487	2.271	3.364	3.685	4.083	4.508	4.907	5.296
ukr_Cyrl	5.115	5.629	5.927	6.315	6.816	7.414	8.085	4.614	5.267	5.618	6.083	6.667	7.317	8.010
urd_Arab	4.746	5.445	5.754	6.098	6.454	6.769	7.004	1.712	4.514	5.140	5.854	6.463	6.925	7.225
uzn_Latn	2.953	3.529	3.786	4.090	4.440	4.797	5.112	2.363	3.181	3.470	3.796	4.132	4.447	4.728
vie_Latn	4.613	4.947	5.083	5.174	5.219	5.239	5.249	2.026	4.168	4.767	5.061	5.146	5.184	5.211
zsm_Latn	3.910	4.409	4.668	4.969	5.296	5.557	5.731	2.594	3.803	4.188	4.621	5.038	5.369	5.573

Table 18: Compression (*bytes per token*) on FLORES for tokenizer extension with continued BPE training. The columns represent different number of added tokens where 0 is the original tokenizer.