Teaching Old Tokenizers New Words: Efficient Tokenizer Adaptation for Pretrained Models

Anonymous ACL submission

Abstract

Tokenizer adaptation plays an important role in transferring pretrained 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**, a simple alternative that extends an existing tokenizer by continuing its merge operations on new data instead of retraining from scratch. 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, and we release an open-source implementation supporting both Byte-level and SentencePiece BPE tokenizers.

1 Introduction

Adapting the vocabulary is a crucial step in tailoring a language model to a new domain through continued pre-training. Most large language models (LLMs) use byte-pair encoding (BPE, Sennrich et al., 2016; Gage, 1994) tokenization. However, the effectiveness of such tokenizers varies greatly across languages, with substantial differences in tokenization efficiency (Petrov et al., 2023), motivating the need for tokenizer adaptation. Modifying the tokenizer — by extending, pruning, or replacing it — offers several benefits. Extending the vocabulary with domain-specific tokens improves compression, reduces sequence lengths, and lowers training and inference costs, with the trade-off of increasing the model size. Pruning infrequent or irrelevant tokens can further reduce model size and memory usage. In all cases where new tokens are added, the model's embeddings have to be re-initialized, which can be done by using the previous tokenizer embeddings for the new tokens (for example FVT, Gee et al., 2022).

In this work, we focus on tokenizer modification of pretrained models through pruning and extending. Extending an existing vocabulary is a widely used step in adapting pre-trained LLMs to new lan-

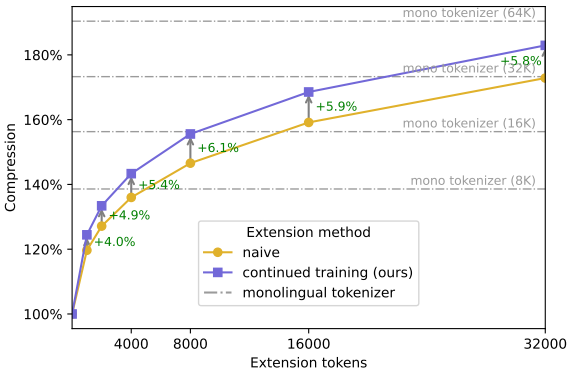

Figure 1: Improvement in compression of Estonian text using the Llama 3 tokenizer extended with Estonian tokens. We compare naive tokenizer extension and continued BPE training (ours). The monolingual tokenizer is trained solely on Estonian data.

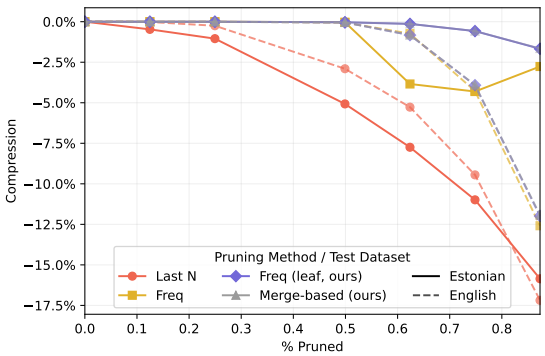

Figure 2: Comparison of pruning methods on Estonian-English pruning (Llama-3).

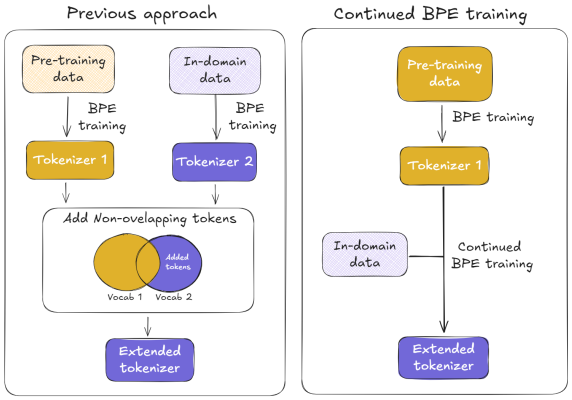

Figure 3: The comparison of vocabulary extension methods: previously common approach of training a tokenizer and using it's vocabulary for extending an existing tokenizer (left) and continuing BPE training on an existing tokenizer proposed by us (right).

guages (Tejaswi et al., 2024; Csaki et al., 2024, 2023; Kiulian et al., 2024; Fujii et al., 2024; Lin et al., 2024; Yamaguchi et al., 2024; Cui et al., 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. We show that this approach introduces useless tokens that do not participate in merges and thus also result in lower compression than having tokens that are useful in tokenization. This happens because BPE strictly follows the merge sequence, e.g., then 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 the training of the original tokenizer on domain-specific data, which accounts for the original merge sequence. We show that for the same number of added tokens, this approach yields better compression (see Figure 1). While we focus on LLMs, this method could also be applied to other types of models that use BPE tokenizers.

The other side of tokenizer vocabulary adaptation, besides extension, is pruning. Thus, we also study the impacts of different kinds of tokenizer pruning and introduce a non-invasive leaf-based pruning that trims the vocabulary in a structure-preserving manner and therefore does not leave unreachable tokens. Furthermore, using the frequencies in a target corpus akin to tokenizer training, pruning allows for preserving the domain- and language-specific tokens required for the new project. This method can be used separately to decrease the vocabulary size, thus largely improving the model's efficiency: for instance, Gemma 3 (Kamath et al., 2025) has the smallest model with 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** (see Figure 2), including a leaf-based pruning algorithm, 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 pretrained models.

2 Methodology

2.1 Vocabulary extension

It is typical 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 (we refer to it as the naive approach). We propose continued tokenizer training for improving the tokenization efficiency (see Figure 3).

We define **continued BPE training** as reapplying the BPE algorithm to in-domain data, using token pair frequencies of the text tokenized with the existing tokenizer to create the new merges. During the creation of the new merges and tokens, we ensure that the token resulting from the merge 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 pretokenizer to segment the text, then tokenizing the resulting segments to count the token pair frequencies within them.

For SentencePiece BPE models (Kudo and Richardson, 2018), we tokenize the text and follow SentencePiece's merging rules by respecting

[1] https://anonymous.4open.science/r/tokenizer-extension

constraints such as keeping different scripts separate and only allowing the space symbol at the beginning of a token. We also extend the vocabulary with characters to ensure vocabulary coverage if necessary, before finding the extension merges, and also ensure that the training data is Unicode NFKC normalized. In both cases, we then run the standard BPE algorithm for tokenizer training using these frequencies, continuing merges until we reach the desired number of new tokens.

While results show that the success of the proposed method depends highly on language, as shown in Section 4, the method itself is language agnostic and similar to BPE tokenizer training, which requires only texts in the target language.

2.2 Vocabulary Pruning

To keep the fixed vocabulary size, we also implement vocabulary pruning. This process involves removing tokens and merges that lead to them in order to decrease the vocabulary size and free the space for vocabulary extension. To determine the tokens that should be pruned first, we try several approaches, based either on token ID 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.

We also introduce a merge-count–based pruning algorithm, which yields results comparable to leaf-based frequency pruning, while avoiding pitfalls of naive pruning methods. As its performance closely matches with leaf-based pruning, we provide a detailed description in Appendix D.

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)
    (t1, t2) ← token_splits[token]
    update t1, t2 in queue by queue[token]
    for t ∈ {t1, t2} :
        downstream_merges[t] −= 1
        if (downstream_merges[t] == 0)
          & (t ∉ atomics) :
            add t to leaves and queue
return pruning_order
```

2.3 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**[2]:

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

The main idea is that if tokenizing a vocabulary token with the same tokenizer does not reproduce that token, then the token cannot be formed through merge operations[3].

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

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

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. When applying this method, the 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.

3 Experimental setup

Datasets. We use Fineweb-2 (Penedo et al., 2025) as the training dataset and set a training budget of 100M characters for the comparison across 70 languages (see Table 9 in Appendix G) 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 A). We evaluate tokenization effectiveness (compression) on the FLORES-200 (Costa-Jussà et al., 2022) dataset using *bytes per token* as the evaluation metric.

We frame the extrinsic evaluation as a case study on Estonian, using Fineweb for English and Fineweb-2 for Estonian as the training data 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 fixed a total budget of 24B characters, allocated equally between Estonian and English. The corpus was then tokenized with the different tokenizers under evaluation, ensuring that the underlying data remained constant across experiments involving tokenizer changes.

Evaluation. For downstream evaluation, we use Estonian and English benchmarks and lm-eval-harness. We evaluate on FLORES-200 (NLLB Team, 2022) (*ET↔EN*), Winogrande (Sakaguchi et al., 2019), Winogrande-ET (Ojastu, 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). For discriminative benchmarks, we report accuracy and 95% confidence interval calculated from the standard error reported by lm-eval-harness. We evaluate FLORES-200 with COMET (`Unbabel/wmt22-comet-da`, Rei et al., 2020, 2022). See Appendix I for more details.

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, which we leverage using the HuggingFace implementation, Llama-2 uses a SentencePiece BPE tokenizer. 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).

4 Results

By continuing tokenizer training 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%) of languages achieving higher compression (see Table 1). For multiple languages, the increase in compression can be more than 20% using continued tokenizer training over the naive method (see Appendix J for full results). This means that during training and inference, we can process the same text using significantly fewer tokens.

We also see a consistent decrease in Rényi entropy (Zouhar et al., 2023), which has been shown to correlate with downstream performance (Zouhar et al., 2023; Arnett and Bergen, 2025).

While our method is language-agnostic, we see that the improvements over extending from an independently trained tokenizer highly depend on the base tokenizer, language, and even how many tokens we are adding. If we look at the average in-

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

	Compression		Renyi entropy		Unreach. added tok	
Model	Δ_{rel}	**WR**	Δ_{rel}	**WR**	**ours**	**naive**
Llama-3						
+1000	3.3%	88.6%	-0.1%	81.4%	0.0%	4.5%
+2000	3.6%	92.9%	-0.6%	95.7%	0.0%	5.7%
+4000	3.8%	98.6%	-1.1%	94.3%	0.0%	7.0%
+8000	3.7%	100.0%	-1.4%	98.6%	0.0%	8.2%
+16000	3.3%	98.6%	-1.3%	98.6%	0.0%	9.3%
+32000	2.7%	92.9%	-1.1%	92.9%	0.0%	10.2%
Qwen-2.5						
+1000	4.3%	98.6%	0.1%	81.4%	0.0%	4.3%
+2000	5.6%	97.1%	-1.0%	92.9%	0.0%	5.6%
+4000	6.9%	100.0%	-1.8%	97.1%	0.0%	7.1%
+8000	8.1%	98.6%	-2.5%	98.6%	0.0%	8.4%
+16000	9.0%	97.1%	-3.0%	98.6%	0.0%	9.6%
+32000	9.6%	94.3%	-3.2%	91.4%	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%	82.86%	0.0%	3.6%
+2000	5.2%	95.7%	-1.1%	91.43%	0.0%	4.8%
+4000	6.1%	90.0%	-1.8%	94.29%	0.0%	6.0%
+8000	6.9%	84.3%	-2.3%	92.86%	0.0%	7.0%
+16000	7.4%	72.9%	-2.7%	82.86%	0.0%	7.8%
+32000	7.7%	74.3%	-2.8%	78.57%	0.0%	8.4%

Table 1: Average scores across 70 languages. We report the **average relative gain** (Δ) and Win Ratio (WR) of continued tokenizer training (ours) over extending from an independently trained tokenizer (naive). Win ratio is the percentage of languages where our method achieves higher scores.

		Qwen-2.5		Llama-3	
script	(n)	**Comp.**	**RE**	**Comp.**	**RE**
Cyrl	(9)	**11.2%**	**-3.3%**	**5.1%**	**-1.8%**
Latn	(35)	9.8%	-2.9%	4.7%	-1.4%
Jpan	(1)	1.1%	-1.2%	1.4%	-0.3%
Arab	(4)	3.8%	-0.3%	0.7%	-0.3%
Hani	(1)	0.2%	-0.2%	-0.0%	0.0%
Other	(20)	2.3%	0.0%	1.2%	0.1%

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

	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 (average across 70 languages) in the heldout target language dataset (n = 10000 documents) calculated through tokenizing the dataset with the extended tokenizer.

crease in compression of our method over the naive method on the Llama-3 and Qwen-2.5 tokenizer, we see it is the most efficient for languages with Cyrillic and Latin scripts (see Table 2).

Why is continued 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 tokenization. To quantify this, we define a metric called *tokens unreachable through merges*. There are more unreachable tokens in extending from a retrained tokenizer (naive) compared to the continued tokenizer training (see Table 1). This means that the naive vocabulary extension method has many useless tokens that will never be used in the merge process. It is important to keep in mind that the Llama-3 benefits from merge skipping (skipping the merging when a pretoken is in the vocabulary), so some of the *tokens unreachable through merges* will be produced in practice.

In addition to calculating the number of tokens we can theoretically not reach through merges, we also tokenized the held-out target language dataset using the Llama-3 tokenizer extended for the target languages. On average, across 70 languages, our method results in 4%-4.7% better utilization of the added token vocabulary in practice (see Table 3).

We hypothesize that if merge skipping is disabled, the improvement is larger. To test this, we disable merge skipping for Llama-3 (enabled by default) and find that for the naive method, the compression drops 1.2–5.8% on average (see Table 8 in Appendix F), depending on the extension size (23.5% at most for a language), while for our method, the difference is less than 0.1% (1.3% at most). This might explain why the difference between our method and the naive method is smaller for Llama-3, which uses merge skipping, compared to Qwen-2.5, which does not.

In comparison of **pruning methods** (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, due

	Size	Continued tokenizer training (ours)					Naive extension				
		ID	ID (L*)	Freq	Freq (L*)	Merge	ID	ID (L*)	Freq	Freq (L*)	Merge
ET	**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	**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

Table 4: Bytes-per-token on FLORES with different pruning methods. In each experiment, we prune the tokenizer 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.

Size	Continued tokenizer training (ours)					Naive extension				
	ID	ID (Leaf)	Freq	Freq (Leaf)	Merge	ID	ID (Leaf)	Freq	Freq (Leaf)	Merge
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 5: Number of unreachable tokens with different pruning methods. In each experiment, we prune the tokenizer using 50%–50% Estonian-English data mix, and then extend it back to the original size using Estonian-only data.

to accounting for the BPE structure. This is better visible in the number of unreachable tokens (Table 5), 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, following the same objective.

4.1 Downstream results

4.1.1 Continued pre-training

Downstream task results for the continually pre-trained Llama-3.2 1B and 3B models in Table 6 show no substantial difference between the naive and continued tokenizer training approaches. In these experiments, we first applied leaf-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 the total training time by 26% owing to the improved compression for Estonian (see Table 10).

In addition to Table 3, which shows that the naive extension method results in lower vocabulary utilization than continued BPE training, we observe signs of undertrained tokens for the naive method after continued pre-training when analyzing embedding changes. A detailed analysis is provided in Appendix C.

Determining when to retain, modify, or replace the tokenizer depends on factors such as model size, dataset size, and the scale of vocabulary extension. Another open question concerns how to account for the reduced number of training tokens in Estonian. These issues are beyond the scope of this work and

Model	Compr. ET (EN)	Toks.	FLORES (COMET) EN-ET	FLORES (COMET) ET-EN	Winogrande (acc) ET	Winogrande (acc) EN	Belebele (acc) ET	Belebele (acc) EN	SIB200 (acc) ET	SIB200 (acc) EN	XCOPA (acc) 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	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	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 6: Downstream task evaluation results on Llama-3 family of models. For tokenizer metrics, see Table 10.

are left for future investigation.

4.1.2 Pruning

We also test pruning methods from the model compression perspective. 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 while aiming to preserve performance in these two languages. Our results in Figure 4 show that with our pruning methods up to 62.5% (80,000) of the tokens can be removed without any noticeable deterioration in downstream performance. When comparing pruning strategies, frequency-leaf 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 see that machine translation (generative) is more sensitive to vocabulary pruning than the discriminative tasks.

5 Related work

Petrov et al. (2023) highlight significant disparities in tokenization efficiency of different languages. Rust et al. (2021) show that monolingual tokenizers outperform multilingual ones for encoder-only models, while Goldman et al. (2024) demonstrate a correlation between compression efficiency and downstream performance. Finally, Arnett and Bergen (2025) suggest that differences in tokenization quality may in small part account for the performance gap between agglutinative and fusional languages. Together, these findings underscore the importance of improving tokenization methods to ensure fair and effective language model performance across languages.

Several works adopted the strategy of training a new tokenizer on domain-specific 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., 2024; Fujii et al., 2024; Lin et al., 2024; Yamaguchi et al., 2024; Cui et al., 2024). Csaki et al. (2023) further propose pruning the least frequent tokens from the original vocabulary before appending new ones to keep the vocabulary size fixed. Unlike prior approaches that extend tokenizers by training a new tokenizer and appending its tokens, our method *continues training* the original tokenizer on domain-specific data. This approach achieves higher compression for the same vocabulary size increase. In addition, we introduce improved pruning techniques that go beyond naive frequency-based methods (Cognetta et al., 2024; Yang et al., 2022), enabling more extensive pruning without degrading compression or downstream performance.

Previous works have also successfully replaced the previous tokenizer with a new one entirely (Samuel et al., 2025). Dagan et al. (2024) show that replacing or significantly altering the tokenizer requires extensive continued pre-training—up to 50B tokens—to avoid performance degradation, and Tejaswi et al. (2024) find that the larger the vocabulary extension, the longer training is needed.

An important aspect of vocabulary extension is the initialization of embeddings for newly added tokens. 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 to-

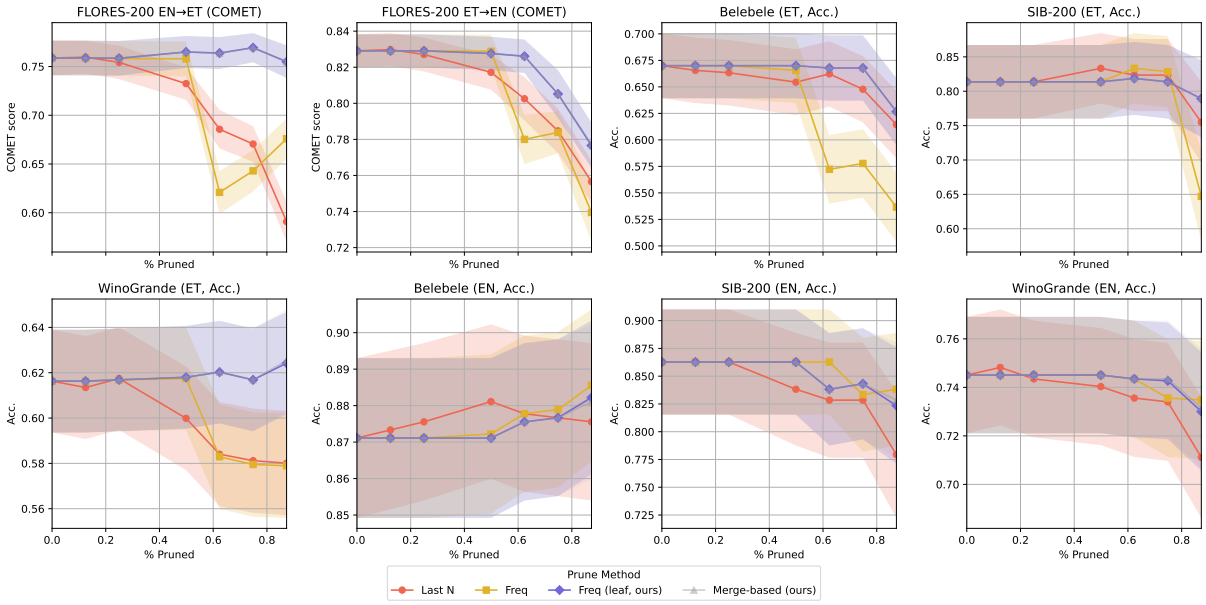

Figure 4: Llama-3.1-8B pruned with different methods and evaluated on Estonian and English downstream tasks. The dataset for training the pruner consisted of 50-50 Estonian-English mix. The highlighted area shows the 95% confidence interval acquired through bootstrap resampling.

kenizer (see Appendix E), performs competitively among initialization strategies. While alternative initialization methods exist (Dobler and de Melo, 2023; Samuel et al., 2025), our work focuses on identifying which tokens to add or prune at the tokenizer level rather than on optimizing embedding initialization (model-level).

6 Discussion

Our method of continued tokenizer training leads to better intrinsic tokenizer performance than the commonly utilized naive extension method. More importantly, it does not harm the BPE structure and leaves no unreachable tokens: this problem can be only partially solved by merge skipping, as it uses only the tokens that constitute complete pre-tokens.

Leaf-based pruning allows for structure-aware tokenizer trimming, which can be used in conjunction with the tokenizer extension or on its own, to reduce the model size, which nowadays is largely dependent on the embedding parameters, especially for the small models. Furthermore, introducing frequencies into this pruning approach makes it possible to remove the vocabulary needed the least, e.g., it is possible to combine a representative sample of all languages or domains needed in the fine-tuned model, not only the ones directly participating in tokenizer extension.

Combining these methods into a library that we release together with this paper makes it a ready-to-use toolkit for adapting tokenizers for any domain and language, while keeping them in the ready-to-use Hugging Face format.

7 Conclusion

We presented methods for controlled vocabulary modification of pretrained 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 improves tokenization efficiency and better 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. We release an open-source toolkit implementing both methods to facilitate further research on vocabulary adaptation for pretrained models.

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 limited to a single language and model, which may affect the generalizability of our findings to other settings.

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.

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.

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 Belhadj Amor, and Grigol Peradze. 2024. From english-centric to effective bilingual: Llms with custom tokenizers for underrepresented languages. *Preprint*, arXiv:2410.18836.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

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: YYYY-MM-DD.

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.

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.

James Cross Onur Çelebi Maha Elbayad Kenneth Heafield Kevin Heffernan Elahe Kalbassi Janice Lam Daniel Licht Jean Maillard Anna Sun Skyler Wang Guillaume Wenzek Al Youngblood Bapi Akula Loic Barrault Gabriel Mejia Gonzalez Prangthip Hansanti John Hoffman Semarley Jarrett Kaushik Ram Sadagopan Dirk Rowe Shannon Spruit Chau Tran Pierre Andrews Necip Fazil Ayan Shruti Bhosale Sergey Edunov Angela Fan Cynthia Gao Vedanuj Goswami Francisco Guzmán Philipp Koehn Alexandre Mourachko Christophe Ropers Safiyyah Saleem Holger Schwenk Jeff Wang NLLB Team, Marta R. Costa-jussà. 2022. No language left behind: Scaling human-centered machine translation.

Marii Ojastu. 2025. Translation of the winogrande benchmark dataset to evaluate commonsense reasoning capabilities of large language models in estonian. Master's thesis, University of Tartu, Tartu, Tartumaa, Estonia, June. Available at `https://dspace.ut.ee/items/d4d729dd-55f7-4371-a032-50558b1c4d94`.

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

Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *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. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

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.

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 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 5). Extending a tokenizer using continued training with 10M character training set yields a bigger effect than extending from a new tokenizer using 10B character training set – the choice of extension method is more important than the training budget at this scale.

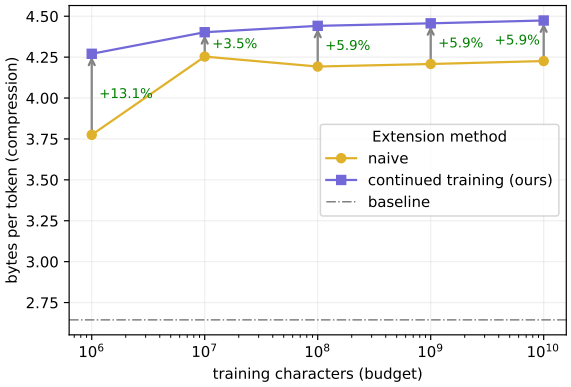

Figure 5: Extension training budget in characters when adding 16,000 tokens to the Llama-3 tokenizer.

B Continued pre-training hyperparameters

The hyperparamters for continued pre-training are in Table 7. 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).

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 7: Continued pre-training hyperparameters.

C Analysis of undertrained tokens

To better understand how the tokenizer extension 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 6 shows that the naive extension method produces noticeably more undertrained tokens, reflected in 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.

D Merge-based pruning

As an alternative to the frequency based pruning with leaf ordering, we also developed the merge-based pruning (see Algorithm 2), which achieves

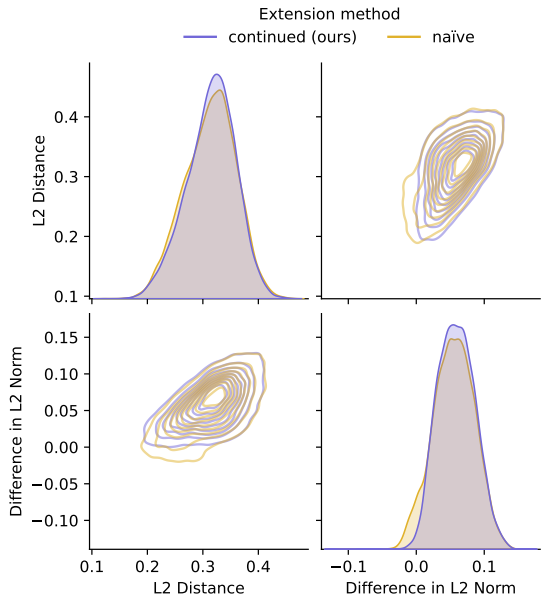

Figure 6: L2 distance of the continually trained and FVT initialized added token input/output embeddings (tied) and the difference in L2 norms of trained vs initialized model of the same tokens (Llama-3.2-3B).

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.

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:

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

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

F The effect of merge skipping

We note that merge skipping contributes significantly to the compression. We look at models that use merge skipping (Llama-3 and Mistral Nemo). We then disable merge skipping and look at how much the performance drops when extending the vocabulary from an auxiliary tokenizer (naive) vs continued BPE training. In Table 8 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 pretoken bypassing merges. As we previously noted the naive method produces many of those unreachable tokens.

G Languages in training data

We provide the overview of the languages in the training dataset in Table 9.

	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 8: 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.

H Continued pretraining tokenizer details

We provide details for the modified Llama-3 tokenizers in the continued pertaining experiments in Table 10.

I Evaluation details

We provide additional evaluation details in Table 11.

J Full tokenizer extension results

We provide the full improvement of our method (continued BPE training) and the naive method in compression and unreachable tokens for Llama-3 and Qwen-2.5 tokenizers in Tables 12 and 13 respectively.

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

			Compression		Renyi Entropy		CPT	
Tokenizer	**|vocab|**	**unreach**	**ET**	**EN**	**ET**	**EN**	**tokens**	**% ET**
default	128k	588	2.64	4.86	0.621	0.485	7.2B	63.9%
prune+ext (16k)	128k	0	4.46	4.86	0.520	0.485	5.3B	50.6%
prune+ext (16k, naive)	128k	2033	4.21	4.86	0.533	0.485	5.4B	52.0%

Table 10: Continued pertaining (CPT) tokenizer overview.

Dataset		Size	Split	n-shots
Winogrande	(Sakaguchi et al., 2019)	1,267	validation	0
Winogrande ET	(Ojastu, 2025)	1,767	test	0
XCOPA	(Ponti et al., 2020)	600	test	5
SIB200	(Adelani et al., 2024)	204	test	5
FLORES200	(NLLB Team, 2022)	1012	devtest	5
Belebele	(Bandarkar et al., 2024)	900	test	5

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

	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 12: 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	-21	-56	-139	-291	-545	-1037	-15	-52	-162	-366	-733	-1442
ary_Arab	-25	-50	-118	-262	-508	-952	-28	-63	-163	-358	-728	-1411
azj_Latn	-148	-339	-830	-1689	-3719	-7716	-137	-309	-751	-1591	-3525	-7402
bel_Cyrl	-43	-118	-296	-681	-1489	-3195	-22	-89	-241	-596	-1330	-2772
ben_Beng	0	0	0	0	-46	-225	0	0	0	-1	-53	-231
bos_Latn	-59	-171	-517	-1446	-3594	-8479	-67	-184	-567	-1534	-3737	-8707
bul_Cyrl	-74	-181	-496	-1288	-3339	-7783	-111	-274	-704	-1751	-4177	-8923
cat_Latn	-60	-133	-342	-752	-1635	-3432	-60	-132	-337	-748	-1623	-3404
ces_Latn	-48	-111	-299	-812	-1974	-4521	-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	-15	-51	-124	-270	-546	-1054	-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	-62	-221	-651	-1686	-4141	-9572	-71	-235	-694	-1780	-4277	-9769
hun_Latn	-75	-174	-411	-1043	-2287	-5143	-73	-167	-404	-1018	-2266	-5054
hye_Armn	0	0	-1	-1	-2	-18	-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	-11	-20	-34	-86	-229	-476	-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	-73	-193	-400	-889	-1953	-3912	-61	-182	-367	-855	-1810	-3790
khk_Cyrl	-68	-152	-387	-910	-1826	-3614	-67	-145	-358	-806	-1644	-3116
khm_Khmr	0	0	0	-3	-18	-119	0	0	0	-2	-23	-136
kir_Cyrl	-68	-152	-394	-927	-1990	-4219	-60	-135	-361	-825	-1774	-3810
kor_Hang	-6	-14	-29	-70	-139	-300	-5	-15	-32	-71	-142	-304
lit_Latn	-71	-177	-454	-1117	-2455	-5304	-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	-85	-253	-612	-1546	-3629	-8263	-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	-566	-1407	-3442	-7897	-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	-4482	-48	-140	-338	-815	-1981	-4499
rus_Cyrl	-37	-88	-197	-413	-873	-1848	-22	-59	-139	-328	-791	-1699
sin_Sinh	-58	-110	-206	-409	-783	-1494	-60	-110	-207	-410	-783	-1494
slk_Latn	-70	-158	-401	-973	-2321	-5343	-52	-152	-408	-990	-2395	-5512
slv_Latn	-87	-222	-592	-1484	-3517	-7891	-73	-210	-595	-1479	-3545	-7863
spa_Latn	-17	-44	-170	-475	-1224	-3033	-18	-44	-165	-473	-1218	-3030
srp_Cyrl	-36	-104	-265	-606	-1506	-3331	-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	-406	0	0	-2	-13	-76	-405
tha_Thai	-7	-13	-26	-52	-103	-248	-2	-7	-34	-70	-136	-334
tur_Latn	-108	-227	-506	-1104	-2447	-5134	-51	-141	-358	-856	-1974	-4377
ukr_Cyrl	-26	-67	-170	-418	-926	-2000	-28	-75	-192	-470	-1008	-2206
urd_Arab	-47	-97	-191	-418	-781	-1391	-53	-121	-310	-596	-1012	-1664
uzn_Latn	-103	-295	-731	-1646	-3508	-7568	-105	-288	-736	-1677	-3577	-7690
vie_Latn	-18	-31	-47	-130	-357	-992	-6	-10	-26	-121	-343	-982
zsm_Latn	-66	-145	-369	-823	-1675	-3153	-65	-141	-360	-802	-1641	-3119

Table 13: **Reduction in unreachable tokens through merges** of **continued tokenizer training** over **naive extension** from an independent tokenizer.