BPE Gets Picky: Efficient Vocabulary Refinement During Tokenizer Training

Pavel Chizhov*[1,2] **Catherine Arnett***[2,3] **Elizaveta Korotkova**[4] **Ivan P. Yamshchikov**[1,2]

[1]CAIRO, Technical University of Applied Sciences Würzburg-Schweinfurt
[2]PleIAs, Paris, France
[3]Department of Linguistics, University of California, San Diego
[4]Institute of Computer Science, University of Tartu

pavel.chizhov@thws.de catherine@pleias.fr
elizaveta.korotkova@ut.ee ivan.yamshchikov@thws.de

Abstract

Language models can greatly benefit from efficient tokenization. However, they still mostly utilize the classical Byte-Pair Encoding (BPE) algorithm, a simple and reliable method. BPE has been shown to cause such issues as under-trained tokens and sub-optimal compression that may affect the downstream performance. We introduce PickyBPE, a modified BPE algorithm that carries out vocabulary refinement during tokenizer training by removing merges that leave intermediate "junk" tokens. Our method improves vocabulary efficiency, eliminates under-trained tokens, and does not compromise text compression. Our experiments show that this method either improves downstream performance or does not harm it.

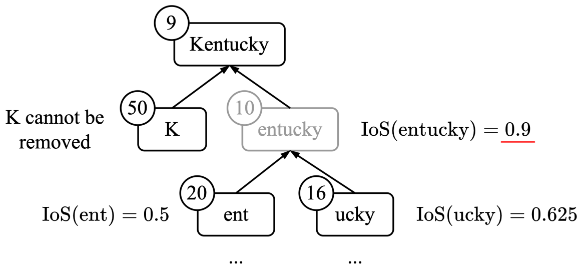


Figure 1: An example of a series of merges to produce the token Kentucky. Pre-merge token frequencies are shown in circles. In the vanilla BPE algorithm, entucky should also be stored in the vocabulary, whereas it is redundant after the merge. In this example, Intersection over Self (IoS) effectively captures the intermediate token, as IoS(entucky) $\geq \mathcal{T} = 0.9$.

1 Introduction

Tokenization is a relatively understudied area, but it can greatly impact model performance and efficiency (Rust et al., 2021; Hofmann et al., 2022; Ali et al., 2024; Toraman et al., 2023; Petrov et al., 2023; Singh and Strouse, 2024; Rajaraman et al., 2024; Shao et al., 2024; Wang et al., 2024). Vocabularies should be efficient, as every additional token in the vocabulary increases the number of embedding parameters, and thus the model size. Each vocabulary item should contribute enough to model performance to justify the use of parameters.

In this paper, we focus on Byte-Pair Encoding (BPE; Gage, 1994; Sennrich et al., 2016) tokenizers. BPE tokenization works by breaking the text down into each of its characters or bytes and then building tokens in the vocabulary through a series of merges. The result of each merge must be stored as a token in the vocabulary. Tokens which are used only to execute merges are sometimes referred to as intermediate "junk" tokens (Bostrom and Durrett, 2020). An example is shown in Figure 1. Intermediate tokens clutter the vocabulary and are hardly ever used during tokenization.

In addition to efficiency, we consider other model behaviors that may be driven by tokenization. Land and Bartolo (2024) recently showed that very low-frequency tokens in the vocabulary may be under-trained by a model. This leads to worse downstream performance and unwanted outputs, such as hallucinations. Under-trained tokens — also called "glitch tokens" (Rumbelow and Watkins, 2023; Geiping et al., 2024; Li et al., 2024) — can also potentially be exploited to circumvent safety measures through the use of these out-of-distribution items.

Vocabulary trimming, which entails removing items from a tokenizer's vocabulary, has been proposed as a method of decreasing the number of unnecessary tokens, *e.g.*, language- or domain-specific tokens. Trimming has been shown to reduce the number of embedding parameters without degrading downstream performance (Ushio et al., 2023; Pang and Vulić, 2024). Under-trained token indicators were shown to be correlated with token frequency in the training corpus, where less frequent tokens are more likely to be under-

*equal contribution

trained (Land and Bartolo, 2024). Vocabulary trimming, thus, is well suited to address the issue of under-trained tokens.

Vocabulary trimming has mostly been implemented as a procedure taking place after tokenizer training (Yang et al., 2022; Cognetta et al., 2024). As a result, it is difficult to determine the vocabulary size in advance, as it is not known beforehand how many tokens will be removed by the trimming procedure. Setting a fixed vocabulary size might be important, for example, for increasing training throughput (Groeneveld et al., 2024).

In this paper, we introduce **PickyBPE**[1] — a modified BPE tokenizer that implements vocabulary refinement during tokenizer training. Unlike other trimming procedures, PickyBPE effectively removes intermediate tokens once they become useless and seamlessly creates a vocabulary of the desired size without data-specific heuristics. Our method leads to more efficient usage of a limited vocabulary and, thus, of the embedding parameters. We show that our method leads to equal or better performance on a downstream translation task compared to standard BPE (Section 4). Furthermore, we reduce the number of tokens that are likely to be under-trained (Section 5). This frees up space for higher-quality tokens. Due to the improved quality of the fixed-size vocabulary, PickyBPE does not compromise text compression (Section 6) unlike other trimming methods, which makes it a good candidate for practical use.

2 Related Work

Several common alternatives to BPE tokenization implicitly address the issue of intermediate low-frequency tokens. For instance, WordPiece tokenization (Wu et al., 2016) is based on a series of merges akin to BPE, but along with the frequency of the token pair being merged, it also takes individual token frequencies into account. Thus, the tokenizer is less likely to add merges that would result in redundant tokens. However, this does not guarantee that the tokenizer adds merges in an optimal order, nor does it facilitate the retrospective removal of intermediate tokens that might eventually appear.

Another popular algorithm is Unigram tokenization (Kudo, 2018) used in SentencePiece (Kudo and Richardson, 2018). The core of this algorithm is different from BPE-like solutions. Unigram works by creating a large vocabulary and iteratively pruning it until it reaches the desired size. The pruning is performed according to how much the token removal affects the likelihood of the subword sequence, and takes into account individual token frequencies. Intermediate tokens are also less likely to appear in such a scenario, which might suggest that Unigram tokenization implicitly performs a form of vocabulary trimming.

There also exist several proposed modifications to BPE, which address the issues raised in Section 1. BPE-Dropout was proposed to mitigate the issue of rare subwords by dropping merges randomly during tokenizer training (Provilkov et al., 2020). This method regularizes BPE training to expose the model to alternate tokenizations of the same strings, making it more robust to noisy input, such as misspellings. BPE-Dropout also helps in reducing the under-training of low-frequency tokens. While it may lead to better training for low-frequency or intermediate tokens, those tokens are never used during model inference. Therefore, this method still leads to unused embedding parameters and ultimately does not improve vocabulary efficiency.

Sennrich et al. (2017) use an absolute frequency cut-off to prevent very low-frequency tokens from being added to the vocabulary. Similarly, Vilar and Federico (2021) propose a stopping criterion in order to select the optimal vocabulary for BPE. The authors propose a maximum likelihood constraint, a point at which BPE stops adding merges during training if a merge decreases the overall likelihood of the token sequence. Building on this work, Cognetta et al. (2024) implement a vocabulary trimming method for BPE. The authors propose removing low-frequency tokens after tokenizer training by manually choosing an absolute threshold value. The choice of threshold, therefore, is specific to a given dataset and vocabulary size. Their method reduces the vocabulary size without significantly reducing downstream translation performance. In some cases, when they show the greatest task improvement, they find an increase of over 13% in sequence length, *i.e.*, text length in number of tokens. Better compression has been shown to correlate with better model performance (Gallé, 2019; Liang et al., 2023; Goldman et al., 2024) and lead to faster inference time (Song et al., 2021; Petrov et al., 2023; Yamaguchi et al., 2024). We argue that a major drawback of the method proposed by Cognetta et al. (2024) is worse compression.

In a concurrent work, Lian et al. (2024) also iden-

[1]https://github.com/pchizhov/picky_bpe

Algorithm 1 PickyBPE Training Step

```
1: Input: Vocabulary $\mathcal{V}$; Tokenized corpus $\mathcal{C}$; Event order $\mathcal{E}$; IoS threshold $\mathcal{T}$
2: Output: Updated $\mathcal{V}, \mathcal{C}, \mathcal{E}$
3: $x_1, x_2 \leftarrow$ the most frequent pair in $\mathcal{C}$
4: $x_3 \leftarrow x_1 + x_2$
5: $\mathcal{V} \leftarrow \mathcal{V} + \{x_3\}$
6: $\mathcal{E} \leftarrow \mathcal{E} + \{\text{Merge}, (x_1, x_2)\}$        ▷ new event
7: if $\text{IoS}(x_1 \mid x_1, x_2) \geq \mathcal{T}$ then
8:     $\mathcal{V} \leftarrow \mathcal{V} \setminus \{x_1\}$        ▷ remove $x_1$
9:     $\mathcal{E} \leftarrow \mathcal{E} + \{\text{Remove}, x_1\}$        ▷ new event
10: end if
11: if $x_2 \neq x_1$ and $\text{IoS}(x_2 \mid x_1, x_2) \geq \mathcal{T}$ then
12:     $\mathcal{V} \leftarrow \mathcal{V} \setminus \{x_2\}$        ▷ remove $x_2$
13:     $\mathcal{E} \leftarrow \mathcal{E} + \{\text{Remove}, x_2\}$        ▷ new event
14: end if
15: Update $\mathcal{C}$ based on events from this iteration
16: return $\mathcal{V}, \mathcal{C}, \mathcal{E}$
```

Algorithm 2 PickyBPE Tokenization

```
1: Input: Word $w$; Vocabulary $\mathcal{V}$; Event order $\mathcal{E}$
2: Output: Tokenized word $\mathcal{W}$
3: $\mathcal{W} \leftarrow$ split $w$ into symbols $\in \mathcal{V}$
4: $\mathcal{M} \leftarrow$ possible merges in $\mathcal{W}$
5: $\mathcal{R} \leftarrow$ possible removals in $\mathcal{W}$
6: while $\mathcal{M} \neq \varnothing$ or $\mathcal{R} \neq \varnothing$ do
7:     $\varepsilon \leftarrow$ earliest event in $\mathcal{E}$, $\varepsilon \in \mathcal{M} \cup \mathcal{R}$
8:     perform $\varepsilon$
9:     update $\mathcal{M}, \mathcal{R}$
10:    exclude events from $\mathcal{E}$ earlier than $\varepsilon$
11: end while
12: return $\mathcal{W}$
```

tify the issue of intermediate ("scaffold") tokens and introduce Scaffold-BPE. The authors propose to identify intermediate tokens when they are below the current range of frequencies during the tokenizer training. Their method does not allow regulating the strength of Scaffold-BPE, unlike our method, which uses a threshold hyperparameter (see Section 3). Additionally, Scaffold-BPE functions by first tokenizing the input text with both the vocabulary and scaffold tokens, then later splitting the scaffold tokens into the minimal valid token sequence. Since this method is a post-processing technique and the final result does not reflect the training process, this inference strategy leads to inaccuracies in tokenization and worse compression (see Appendix A).

Another contemporaneous work by Bauwens and Delobelle (2024) also proposes a method of pruning merges that lead to undesired segmentation and underused vocabulary items. This method differs in two key ways from previous approaches. It allows merges of more than two tokens and uses a semi-supervised method to determine which merges to remove based on manually annotated language-specific morphological segmentations. The latter makes it hard to generalize this method to other languages, especially in multilingual settings.

3 PickyBPE

Our method is a modification of the original BPE algorithm (Gage, 1994; Sennrich et al., 2016). The intuition behind PickyBPE is that we can identify intermediate tokens based on their individual frequency and frequency within a larger token. Intermediate tokens should have low frequency outside of the context of the token that contains them. For example, in Figure 1, an intermediate token `entucky` is almost always a part of `Kentucky`, which is easy to capture by comparing the frequencies of `Kentucky` and `entucky`. To formalize this approach, we introduce a measure called *Intersection over Self (IoS)*, which is computed as follows:

$$\text{IoS}(x_1 \mid x_1, x_2) = \frac{f_p(x_1, x_2)}{f_t(x_1)}; \qquad (1)$$

$$\text{IoS}(x_2 \mid x_1, x_2) = \frac{f_p(x_1, x_2)}{f_t(x_2)}. \qquad (2)$$

Here x_1 and x_2 are the tokens being merged, f_t is token frequency, and f_p is pair frequency. $\text{IoS}(x_1 \mid x_1, x_2)$ shows how often token x_1 occurs as part of a pair $\{x_1, x_2\}$ compared to all occurrences of x_1. If this value is high, *i. e.*, close to 1, x_1 is highly likely an intermediate token, an integral part of a longer, more meaningful token $x_1 + x_2$. Adding $x_1 + x_2$ to the vocabulary makes x_1 redundant and we can consider removing it.

3.1 Algorithm

The training of PickyBPE follows the main steps of the vanilla BPE training. The text is first split into a sequence of characters/bytes, initializing the vocabulary with unique symbols. Optionally, the coverage parameter (we use 0.9999 in our experiments) is used to replace the rarest symbols with `<unk>`. After that, the algorithm iteratively chooses the most frequent pair of tokens to merge and adds it to the vocabulary. PickyBPE diverges from vanilla BPE in that after each merge when we check

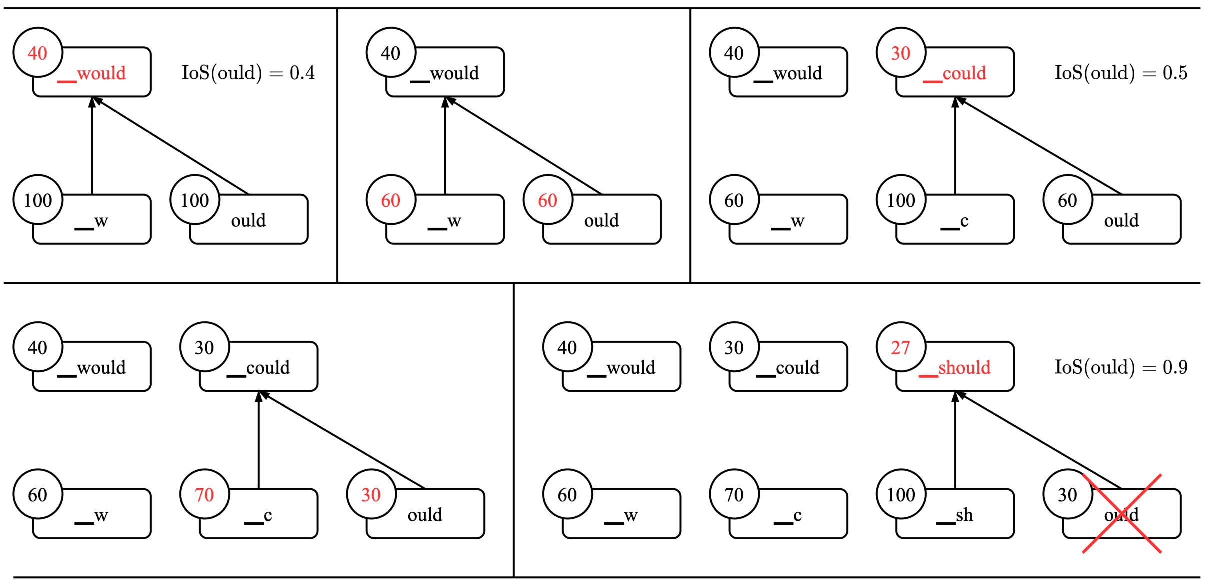


Figure 2: Example of PickyBPE tokenizer training. Token frequencies are shown in the corresponding circles and are updated on merges. Token "ould" is removed only after merging into three common tokens containing it. The corresponding IoS values are visualized on every merge. Once IoS becomes greater than or equal to the threshold $\mathcal{T}$ (0.9 in this example), the token "ould" is removed.

whether we can remove any of the merged tokens according to its IoS value. The pseudocode for a training step is demonstrated in Algorithm 1. We integrate the IoS metric into the merging stage. When a pair of tokens is merged, we check whether we can safely remove either of the two tokens from the vocabulary. For this, we introduce a hyperparameter $\mathcal{T}$, the IoS threshold. If $\text{IoS}(x_1 \mid x_1, x_2) \geq \mathcal{T}$, we remove x_1. Thus, $\mathcal{T}$ regulates the strength of the removal policy: $\mathcal{T}$ is a positive value ≤ 1, and the closer it is to 1, the less strict the removing criterion becomes. For instance, $\mathcal{T} = 0.9$ means that only the tokens that occur outside of the new merge in no more than 10% of cases will be removed. In an extreme case, $\mathcal{T} = 1$ means that no removals are possible, thus the algorithm becomes the vanilla BPE. Another unique feature of our algorithm is that the merges and removals are stored in the event order array $\mathcal{E}$ in the chronological order. Preserving the original order of events is crucial for the tokenization step.

The tokenization (inference) step is described in Algorithm 2. We first split the input word into a series of in-vocabulary symbols. Then we collect the sets of possible merges and removals in the current tokenization and iteratively greedily choose the earliest possible event using event order $\mathcal{E}$. The action associated with the chosen event is performed and the sets of possible merges and removals are updated. This process strictly follows the tokenizer training and avoids compression issues that occur with approximation methods (see Appendix A).

3.2 Algorithm analysis and justification

The training of PickyBPE takes longer than that of the original vanilla BPE. However, the difference is not drastic. When a token is removed, recalculating the frequencies requires a constant number of operations, which makes the training time depend linearly on the number of events (merges and removals). With threshold $\mathcal{T}$ values of 0.6 and higher, the proportion of removed tokens generally does not surpass 10% (for details refer to Appendix E), which makes the number of removals inferior to the number of merges. At the tokenization stage, the inference time depends on the number of events, just as the tokenization time of the vanilla BPE depends on the number of merges. As we show in Appendix E, merges comprise the largest proportion of overall events. Therefore, the removal events do not significantly slow down the inference.

In addition to achieving the primary goal of removing intermediate tokens from the vocabulary, our algorithm has several useful inherent properties. Below, we describe the most notable of those.

Universal threshold. The threshold $\mathcal{T}$ is relative and does not depend on the size of the training corpus or the desired vocabulary. This is one of the advantages of our method compared to the main counterparts, such as Cognetta et al. (2024). Furthermore, the removals happen during training, yielding a vocabulary of the desired size that does not require any post-processing.

Variety of intermediate tokens. An intermediate token may be used to form more than one new token, as shown in Figure 2. Our algorithm handles these cases, removing the token only after there are few to no words it can be merged into.

Second chances. Any removed token may be added again if it becomes the most frequent at a later point in the order of merges. This is usually the case for tokens removed in the very beginning when the frequencies of new tokens are very high. For example, (`"t"`, `"he"`) is likely to be merged early in tokenizer training because `"the"` is a frequent word. Since the relative frequency of `"he"` is lower, `"he"` may be split into (`"h"`, `"e"`). But as `"he"` is still a high-frequency word, it is likely to be merged again. If a previously removed token is restored, it is re-activated to keep its original place in the list of merges. This is essential to the merge order during tokenization.

4 Machine Translation Experiments

To evaluate the downstream performance of our algorithm, we conduct several machine translation (MT) experiments. We train MT models for three translation directions: English–German (EN–DE), German–Estonian (DE–ET), and Ukrainian–Estonian (UK–ET).[2] With this choice of language pairs, we aim to cover diverse MT tasks of varying difficulty. German and English are related languages and share the same script. This language pair represents an easier translation task. German and Estonian use the same script, but are much less closely related, belonging to different language families. Translation for this pair should be more difficult. Finally, Ukrainian and Estonian represent the most difficult translation pair in our experiments. These languages are not only distant but also use different scripts.

To train the EN–DE models, we use the training corpus from the WMT16 news translation task (Bojar et al., 2016), with `newstest2016` corpus for evaluation. For DE–ET and UK–ET, we use the mixtures of parallel corpora assembled by Korotkova and Fishel (2024). For the evaluations of outputs in Estonian, we use the development set of the FLORES benchmark (Goyal et al., 2022).

[2]We also experiment with a different type of writing system on the example of Estonian–Chinese (ET–ZH). Results and discussion of these experiments are presented in Appendix C.

Experiment	$\mathcal{T}$	BLEU (↑)	COMET (↑)
EN–DE	1.0*	30.1 ± 0.7	0.431
	0.9	30.3 ± 0.7	0.431
	0.8	30.0 ± 0.7	0.431
	0.7	30.6 ± 0.7	**0.434**
	0.6	30.3 ± 0.7	0.431
DE–ET	1.0*	19.4 ± 1.0	0.516
	0.9	19.9 ± 1.0	**0.520**
	0.8	19.8 ± 1.0	**0.520**
	0.7	19.9 ± 1.0	**0.520**
	0.6	19.9 ± 1.1	**0.520**
UK–ET	1.0*	16.9 ± 1.0	0.506
	0.9	15.8 ± 1.5	0.508
	0.8	16.7 ± 1.3	**0.511**
	0.7	17.2 ± 1.0	0.509
	0.6	16.9 ± 0.9	**0.511**

Table 1: Machine translation results with vocabulary size 8192 on `newstest2016` set (Bojar et al., 2016) for EN–DE, and on FLORES-dev (Goyal et al., 2022) for DE–ET and UK–ET. For every threshold $\mathcal{T}$, we report BLEU (Papineni et al., 2002) and COMET (Rei et al., 2020) scores. The best scores are highlighted in **bold**. Other scores that are not statistically significantly different from the best are also highlighted in **bold**. If none of the scores are significantly better than the rest, nothing is highlighted. $^*\mathcal{T} = 1.0$ represents the baseline vanilla BPE without intermediate token removal.

We test our method with several thresholds: 0.6, 0.7, 0.8, and 0.9. We did not consider lower thresholds as they would remove too many useful tokens. For the baseline, we chose vanilla BPE, which we obtained by training our PickyBPE with $\mathcal{T} = 1$ to ensure that the effects are not driven by implementation differences. We use the `transformer-iwslt` model from `fairseq` (Ott et al., 2019) for all MT tasks. The architecture and training details can be found in Appendix B.

For generation, we use beam search with beam size 5 in all our experiments. We use BLEU (Papineni et al., 2002) from `sacreBLEU` (Post, 2018) and COMET (Rei et al., 2020) scores for automatic evaluation. We compute paired t-Test with bootstrapping[3] to test for statistical significance on performance metrics (Koehn, 2004).

[3]We evaluate 1000 bootstrap resamples and use t-Test with confidence level 0.95.

Smaller vocabularies. First, we conduct experiments on all three language pairs with a small vocabulary size of 8192. We chose such a restrictive setting to make sure all the tokens are sufficiently trained, as the relatively small training datasets we used (∼1–4M sentence pairs) do not necessitate large vocabularies (Sennrich and Zhang, 2019). The results are presented in Table 1. Overall, the models trained with PickyBPE vocabulary performed comparably to the vanilla BPE, with at least one PickyBPE threshold significantly outperforming it for all three translation directions according to the COMET metric. COMET scores for the DE–ET experiment show that all PickyBPE models were better than the vanilla baseline.

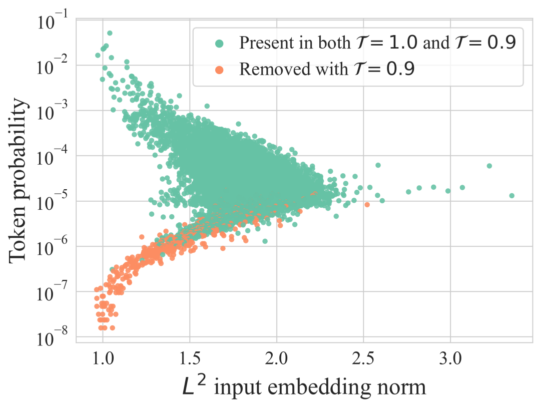
Larger vocabularies. We also tested PickyBPE with larger vocabularies for the EN–DE task. We used two settings: separate vocabularies for input and output, and joint vocabularies. In both cases, we used total vocabulary sizes 16384, 32768, and 65536. The results for these experiments are presented in Table 2. As with the smaller vocabulary setting, we see models based on PickyBPE tokenization performing at least on par with the ones based on the vanilla BPE. In most experiments, our method brings downstream improvements, as shown by the values of the COMET metric. We also observe by the BLEU scores that for the largest vocabularies of sizes 32768 + 32768 and 65536 the performance is generally worse than with the smaller vocabularies, regardless of the tokenization method. This is likely due to the volume of training data being insufficient for such a large vocabulary. However, in this setting PickyBPE still outperforms vanilla BPE according to COMET.

5 Under-Trained Tokens

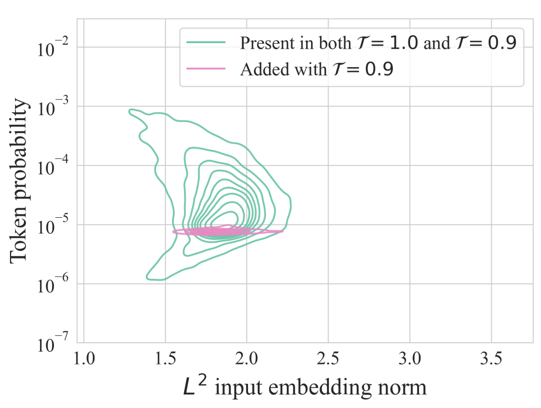
We also test whether PickyBPE decreases the number of tokens likely to be under-trained. These tokens can be identified by looking for very low L^2 norm of the token embeddings (Land and Bartolo, 2024). We plot L^2 norms for $\mathcal{T} = 0.9$ in Figure 3 and those for the remaining thresholds in Appendix D. There are two groups of low-L^2 norm tokens: the first is the low-frequency tokens, which can be seen in the lower left of Figure 3a. According to Land and Bartolo (2024), this might indicate under-training. There is also a group of the highest-frequency tokens with low L^2 norms (top left of Figure 3a). We posit that these are general-purpose tokens that occur in a wide variety of contexts, and thus their representations are less specific. It has long been observed that high-frequency words are

Vocabulary	$\mathcal{T}$	BLEU (↑)	COMET (↑)
8192 + 8192	1.0*	30.7 ± 0.7	0.431
	0.9	30.4 ± 0.7	0.431
	0.8	30.3 ± 0.7	0.430
	0.7	30.3 ± 0.7	0.430
	0.6	30.8 ± 0.7	**0.432**
16384 + 16384	1.0*	31.1 ± 0.7	0.433
	0.9	31.1 ± 0.7	0.433
	0.8	31.0 ± 0.7	**0.435**
	0.7	31.4 ± 0.7	**0.435**
	0.6	31.1 ± 0.7	**0.435**
32768 + 32768	1.0*	29.8 ± 0.7	0.418
	0.9	29.6 ± 0.8	**0.428**
	0.8	30.5 ± 0.7	**0.430**
	0.7	30.4 ± 0.7	**0.430**
	0.6	28.3 ± 0.8	0.416
16384	1.0*	31.1 ± 0.7	0.436
	0.9	31.2 ± 0.7	0.436
	0.8	30.9 ± 0.6	0.434
	0.7	31.1 ± 0.7	0.436
	0.6	31.3 ± 0.7	**0.438**
32768	1.0*	30.9 ± 0.7	**0.435**
	0.9	31.1 ± 0.7	0.434
	0.8	31.1 ± 0.7	**0.437**
	0.7	30.9 ± 0.7	**0.436**
	0.6	30.9 ± 0.7	0.431
65536	1.0*	28.5 ± 0.7	0.421
	0.9	28.4 ± 0.7	**0.427**
	0.8	28.6 ± 0.7	**0.425**
	0.7	28.0 ± 0.7	0.416
	0.6	28.8 ± 0.7	0.420

Table 2: Machine translation results on EN–DE `newstest2016` set (Bojar et al., 2016) with larger vocabularies: 8192, 16384, and 32768 for each language separately, and joint vocabularies of sizes 16384, 32768, and 65536. For every threshold $\mathcal{T}$, we report BLEU (Papineni et al., 2002) and COMET (Rei et al., 2020) scores. The best scores are highlighted in **bold**. Other scores that are not statistically significantly different from the best are also highlighted in **bold**. If none of the scores are significantly better than the rest, nothing is highlighted. $^*\mathcal{T} = 1.0$ represents the baseline vanilla BPE without intermediate token removal.



(a) PickyBPE tokens when $\mathcal{T} = 1.0$. The tokens that are present when $\mathcal{T} = 1.0$ but removed when $\mathcal{T} = 0.9$ (orange) are generally infrequent and have low L^2 embedding norms, thus the majority of them are likely to be under-trained (Land and Bartolo, 2024).



(b) PickyBPE tokens when $\mathcal{T} = 0.9$. The tokens that are present when $\mathcal{T} = 0.9$ but not when $\mathcal{T} = 1.0$ (pink) have frequencies and L^2 norms of the embeddings close to the blob center and thus are less likely to be under-trained (Land and Bartolo, 2024).

Figure 3: Input embedding vectors for PickyBPE tokens with **(a)** $\mathcal{T} = 1$ and **(b)** $\mathcal{T} = 0.9$ for English vocabularies of size 16384 in EN–DE experiments with separate vocabularies. For each token we compute its probability in the training corpus (y-axis), and the L^2 norm of its embedding vector in the trained model (x-axis).

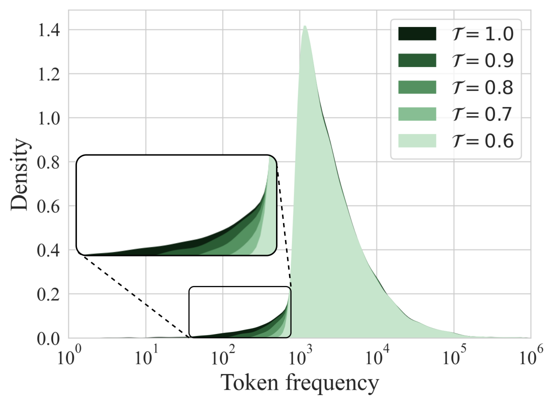


Figure 4: Token frequency distributions for English vocabularies of size 16384 in EN–DE experiments with separate vocabularies for input and output. The left tail becomes less heavy as we decrease the threshold.

$\mathcal{T}$	# unique tokens vs vanilla BPE	# unique tokens vs vanilla BPE + post-trimming
0.9	168 (2.1%)	115 (1.4%)
0.8	391 (4.8%)	248 (3.0%)
0.7	625 (7.6%)	393 (4.8%)
0.6	869 (10.6%)	588 (7.2%)

Table 3: Comparison of tokens from PickyBPE and vanilla BPE for joint EN–DE vocabularies of size 8192. For each threshold $\mathcal{T}$, we report the number of tokens that are present in the PickyBPE but not in the vanilla BPE ($\mathcal{T} = 1$) vocabulary with and without low-frequency token trimming on post-processing.

more likely to have more senses, *i.e.*, meanings (Zipf, 1945), and thus be more general-purpose.

A large portion of tokens removed by PickyBPE (Figure 3a) are likely to become under-trained. By contrast, the new tokens added by PickyBPE (Figure 3b) have higher L^2 norms and higher probability of occurrence. The high-frequency general tokens are not removed by PickyBPE. We argue that PickyBPE reduces the likelihood of under-trained tokens and thus the risks associated with them, such as increased hallucinations.

We also find that as we lower the threshold for PickyBPE, there is a decrease in the left tail of the token frequency distribution, which represents the low-frequency tokens (Figure 4). Trimming methods that involve an absolute frequency cutoff, such as the one used by Cognetta et al. (2024), would completely eliminate the left tail and leave an abrupt drop in the distribution. We observe that PickyBPE preserves the overall distribution and does not eliminate the left tail. This shows that PickyBPE is fundamentally different from post-training trimming of low-frequency tokens.

Table 3 shows the difference in the number of tokens present in the PickyBPE but not in the vanilla BPE vocabulary obtained with and without post-trimming. By post-trimming we mean training the vanilla BPE to have a larger vocabulary with further trimming of low-frequency tokens to achieve the desired vocabulary size. To obtain vocabularies

Threshold	# removed	Compression (↓)		% Word-Initial Tokens			Mean Token Length (↑)
		German	English	Dropped (↓)	Added (↑)	Overall (↑)	
1.0*	0	1.000	1.000	—	—	61.5	5.38
0.9	160	0.997	0.996	43.8	65.5	61.9	5.40
0.8	358	0.995	0.993	**41.1**	**67.5**	62.7	5.44
0.7	588	0.994	0.991	42.0	66.9	63.3	5.47
0.6	805	**0.992**	**0.989**	42.1	64.2	**63.6**	**5.50**

Table 4: Tokenizer evaluation on EN–DE tokenizers with joint vocabularies of size 8192. Compression is reported as corpus token counts of the `newstest2016` set relative to the vanilla BPE. 1 indicates the same compression rate. We report the proportion of word-initial tokens out of dropped tokens (the ones present in vanilla BPE, but not in PickyBPE), added tokens (the ones present in PickyBPE, but not in vanilla BPE), and out of the whole vocabulary along with the mean token length in characters. $^*\mathcal{T} = 1.0$ represents vanilla BPE.

of equal sizes for a fair comparison, we train the initial vanilla BPE tokenizer so that the number of additional tokens reaches the number of replaced tokens from the corresponding PickyBPE tokenizer. Through the differences in numbers of replaced tokens, we show that PickyBPE is not simply a different implementation of the post-trimming akin to Cognetta et al. (2024), but leads to a fundamentally different resulting vocabulary.

6 Features of PickyBPE

Text Compression. Text compression is generally considered to be an important aspect of tokenizer evaluation (Gallé, 2019; Goldman et al., 2024), and language models that compress more have been shown to perform better (Liang et al., 2023; Goldman et al., 2024). We use *corpus token count* (CTC; Schmidt et al., 2024) to measure compression. CTC, also called sequence length, is the number of tokens needed to represent a given text. The fewer tokens are needed, the better the compression.

Table 4 shows the changes in compression as a percentage relative to the tokenizer of the same vocabulary size with a threshold of 1, all for EN–DE vocabularies of size 8192. We report additional compression rates in Appendix F. We find that PickyBPE shows no loss in compression. This is an improvement over the method in Cognetta et al. (2024), which shows worse compression after vocabulary trimming.

Token Qualities. In addition to the above metrics, we compare the tokens themselves. One quality of interest is the proportion of word-initial tokens, which are stored in the tokenizer with an underscore at the beginning to represent a space character. Yehezkel and Pinter (2023) also notice that their trimming procedure leads to an increased number of word-initial tokens.

In Table 4, we also report the percentage of word-initial tokens among the added and removed tokens as well as overall proportions for the EN–DE vocabulary of size 8192. We report results for the other experiments in Appendix G. We find that dropped tokens are far less likely to be word-initial than added tokens. Therefore, PickyBPE is adding more word-initial tokens than it is removing. As the threshold is lowered, we see slightly fewer word-initial tokens added to the vocabulary. This might be due to the intensive removals happening with lower thresholds. In the overall rates of word-initial tokens, we see a slight increase as $\mathcal{T}$ goes down.

Upon inspection of the added tokens, we see that many of the word-initial tokens are also complete words, for example `_renovated`, `_overcoat`, `_cognition`, and `_unconventional`. Increased rates of word-initial tokens may be indicative of improved token quality.

Many of the tokens removed by PickyBPE were intermediate, much like `entucky` (Figure 1). These tokens are relatively long and only occur in the context of a longer token that is also present in the vocabulary. Often, these tokens are missing only one or two characters compared to the full word. We find word-initial and word-medial intermediate tokens, *e.g.*, `_Chicag`, `_algorith`, `roprietary`, `omenclature` (*cf.* ‘Chicago’, ‘algorithm’, ‘proprietary’, ‘nomenclature’).

Following Bostrom and Durrett (2020), we also measure mean token length. They argue that longer mean token length is associated with gold-standard morphologically-aligned tokenization, and thus

Method	CTC (↓)		% Word-initial (↑)	Mean len (↑)
	EN	DE		
Unigram	1.143	1.124	**75.6**	**7.73**
$\mathcal{T} = 1.0$	1.000	1.000	72.2	6.85
$\mathcal{T} = 0.9$	0.997	0.998	72.8	6.88
$\mathcal{T} = 0.8$	0.996	0.998	73.2	6.91
$\mathcal{T} = 0.7$	0.994	0.997	73.6	6.94
$\mathcal{T} = 0.6$	**0.992**	**0.996**	73.9	6.95

Table 5: Comparing PickyBPE and Unigram (Kudo, 2018) tokenizers on joint EN–DE vocabularies of size 32768. We report compression as corpus token counts (CTC) on the `newstest2016` set relative to those of the vanilla BPE ($\mathcal{T} = 1.0$), percentage of word-initial tokens, and mean token length in characters (denoted in the table by "Mean len").

with better token quality. Additionally, longer tokens on average will lead to increased compression, as a text of a fixed length can be represented with fewer longer tokens. We find that the mean token length slightly, but consistently, increases as we lower the IoS threshold (see Table 4). We report additional mean token length results in Appendix H.

We also compare PickyBPE with Unigram tokenization in Table 5. Unigram tokenization seems to yield longer tokens with a higher proportion of word-initial tokens. However, it drastically worsens the compression. We hypothesize that Unigram adds many meaningful full-word tokens which are not optimal for the text compression under the restriction of the vocabulary size.

7 Discussion

We provide a series of experiments that illustrate key properties of PickyBPE. To put these results into perspective, we wish to reiterate two core aspects of the provided experiments: first, there is no universal methodology that could assess tokenizer quality; second, the inefficiencies associated with under-trained tokens discussed by Land and Bartolo (2024) depend on the size of vocabulary relative to the size of training data.

Evaluating tokenizers. It is not always clear how to best compare different tokenizers (Zouhar et al., 2023). One approach is training models for each tokenizer and evaluating downstream performance, *e.g.*, Goldman et al. (2024). However, these results may be driven by confounding factors, such as differences in compression leading to the model effectively being trained on less text (Petrov et al., 2023), and downstream results may also be task-specific. The second general approach to evaluating tokenizers is to evaluate some quality of the tokenizer's output such as compression, similarity of tokenizer boundaries to morphological boundaries (Hofmann et al., 2021), or cognitive plausibility of tokens (Beinborn and Pinter, 2023). There is no consensus about which metric(s) provide the best overall estimation of tokenizer quality.

Role of under-trained tokens. We achieved better or equal performance on machine translation with small vocabularies compared to the vanilla BPE, which suggests that removing under-trained tokens benefits downstream performance. With larger vocabularies, we also saw improvement from our method judging by the values of the COMET metric. However, the largest vocabulary sizes we experimented with were too large for the rather small datasets we used, which led to degraded overall performance regardless of the tokenizer choice. To see the fair improvement with larger vocabulary sizes, one might consider testing PickyBPE as a tokenizer for an LLM, where vocabulary size will be significantly scaled up together with the size of the training data. In this scenario, we also expect to see the benefits of removing under-trained tokens, which were investigated by Land and Bartolo (2024) specifically on the example of LLMs. We leave this exploration for future work.

8 Conclusion

In this paper, we propose PickyBPE, a novel tokenization algorithm that refines the vocabulary during tokenizer training by removing intermediate tokens. Our machine translation experiments show that this algorithm either matches or surpasses the downstream performance shown by vanilla BPE on the same data, which we can extrapolate to larger vocabularies and data given enough training. Our method also mitigates the issue of under-trained tokens, efficiently removing them during tokenizer training, and improves token quality and text compression, filling the freed vocabulary space with higher-frequency and higher-quality tokens. These factors suggest that PickyBPE can be considered for larger models, *e.g.*, LLMs, to improve downstream performance and safety and avoid undesired behavior, *e.g.*, hallucinations.

9 Limitations

PickyBPE behavior depends on the choice of threshold $\mathcal{T}$. Even though the threshold is relative and mostly intuitive in use, one must consider that with lower thresholds the probability of eliminating useful tokens grows and the behavior becomes less stable. Therefore, it is important to start with safer larger thresholds, analyzing the tokenization using token quality measures.

In this paper, the only downstream task we evaluate our models on is translation. Training a larger language model and evaluating it on other downstream tasks may show different patterns. This may allow us to better understand the contribution of PickyBPE as well as its potential drawbacks.

Rust et al. (2021) show that different tasks have variable correlation with tokenizer evaluations like fertility. To the best of our knowledge, there is not enough empirical work to determine which tasks would be most informative for evaluating tokenizer quality. This is an important area for future work.

Our experiments are also limited to a relatively small set of languages. We selected pairs of languages that were typologically varied and used different writing systems, however, most of the languages are spoken in Europe. Our experiments with Chinese did not show any improvements in performance from PickyBPE, which we argue is due to the inherent properties of the writing system (see Appendix C for discussion). In future work, a broader sample of languages could be covered to verify the consistency of the observed trends.

Acknowledgments

The authors acknowledge the HPC resource allocation by Erlangen National High-Performance Computing Center (NHR@FAU) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) (joint project with the Center for Artificial Intelligence (CAIRO), THWS) and Jean Zay (compute grant #GC011015451). The authors would also like to thank the other members of the PleIAs Research team for helpful discussion and feedback.

References

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

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.

Lisa Beinborn and Yuval Pinter. 2023. Analyzing cognitive plausibility of subword tokenization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4478–4486, Singapore. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Kaj Bostrom and Greg Durrett. 2020. Byte Pair Encoding is Suboptimal for Language Model Pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

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.

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

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

Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. 2024. Coercing

LLMs to do and reveal (almost) anything. *arXiv preprint arXiv:2402.14020*.

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 and virtual meeting. Association for Computational Linguistics.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. 2022. The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. 2024. OLMo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809, Bangkok, Thailand. Association for Computational Linguistics.

Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. 2021. Superbizarre is not superb: Derivational morphology improves BERT's interpretation of complex words. 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 3594–3608, Online. Association for Computational Linguistics.

Valentin Hofmann, Hinrich Schuetze, and Janet Pierrehumbert. 2022. An Embarrassingly Simple Method to Mitigate Undesirable Properties of Pretrained Language Model Tokenizers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 385–393, Dublin, Ireland. Association for Computational Linguistics.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Elizaveta Korotkova and Mark Fishel. 2024. Estonian-Centric Machine Translation: Data, Models, and Challenges. In *Proceedings of the 25th Annual Conference of the European Association for Machine Translation (Volume 1: Research And Implementations & Case Studies)*, pages 647–660, Sheffield, UK. European Association for Machine Translation.

Taku Kudo. 2018. Subword Regularization: Improving Neural network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

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

Sander Land and Max Bartolo. 2024. Fishing for Magikarp: Automatically Detecting Under-trained Tokens in Large Language Models. *arXiv preprint arXiv:2405.05417*.

Yuxi Li, Yi Liu, Gelei Deng, Ying Zhang, Wenjia Song, Ling Shi, Kailong Wang, Yuekang Li, Yang Liu, and Haoyu Wang. 2024. Glitch tokens in large language models: Categorization taxonomy and effective detection. *Proc. ACM Softw. Eng.*, 1(FSE).

Haoran Lian, Yizhe Xiong, Jianwei Niu, Shasha Mo, Zhenpeng Su, Zijia Lin, Peng Liu, Hui Chen, and Guiguang Ding. 2024. Scaffold-bpe: Enhancing byte pair encoding with simple and effective scaffold token removal. *arXiv preprint arXiv:2404.17808*.

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

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jiayun Pang and Ivan Vulić. 2024. Specialising and Analysing Instruction-Tuned and Byte-Level Language Models for Organic Reaction Prediction. *arXiv preprint arXiv:2405.10625*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. 2024. Toward a Theory of Tokenization in LLMs. *arXiv preprint arXiv:2404.08335*.

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.

Jessica Rumbelow and Matthew Watkins. 2023. SolidGoldMagikarp (plus, prompt generation).

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization Is More Than Compression. *arXiv preprint arXiv:2402.18376*.

Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. 2017. The University of Edinburgh's neural MT systems for WMT17. In *Proceedings of the Second Conference on Machine Translation*, pages 389–399, Copenhagen, Denmark. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich and Biao Zhang. 2019. Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy. Association for Computational Linguistics.

Ninglu Shao, Shitao Xiao, Zheng Liu, and Peitian Zhang. 2024. Flexibly Scaling Large Language Models Contexts Through Extensible Tokenization. *arXiv preprint arXiv:2401.07793*.

Aaditya K Singh and DJ Strouse. 2024. Tokenization counts: the impact of tokenization on arithmetic in frontier LLMs. *arXiv preprint arXiv:2402.14903*.

Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. 2021. Fast WordPiece Tokenization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2089–2103, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2023. Impact of tokenization on language models: An analysis for turkish. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 22(4).

Asahi Ushio, Yi Zhou, and Jose Camacho-Collados. 2023. Efficient Multilingual Language Model Compression through Vocabulary Trimming. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14725–14739, Singapore. Association for Computational Linguistics.

David Vilar and Marcello Federico. 2021. A statistical extension of byte-pair encoding. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 263–275, Bangkok, Thailand (online). Association for Computational Linguistics.

Dixuan Wang, Yanda Li, Junyuan Jiang, Zepeng Ding, Guochao Jiang, Jiaqing Liang, and Deqing Yang. 2024. Tokenization Matters! Degrading Large Language Models through Challenging Their Tokenization. *arXiv preprint arXiv:2405.17067*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Atsuki Yamaguchi, Aline Villavicencio, and Nikolaos Aletras. 2024. An Empirical Study on Cross-lingual Vocabulary Adaptation for Efficient Generative LLM Inference. *arXiv preprint arXiv:2402.10712*.

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.

Shaked Yehezkel and Yuval Pinter. 2023. Incorporating Context into Subword Vocabularies. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 623–635, Dubrovnik, Croatia. Association for Computational Linguistics.

George Kingsley Zipf. 1945. The meaning-frequency relationship of words. *The Journal of general psychology*, 33(2):251–256.

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 Inference options

PickyBPE inference strictly follows the training order of events and executes merges and removals in the same chronological order (Algorithm 2). Concurrent works use a different approach to inference: the input text is first tokenized with a vanilla BPE tokenizer using both active and removed tokens and then the low-frequency (Cognetta et al., 2024) or scaffold (Lian et al., 2024) tokens are split into the shortest available sequences of valid tokens. The latter approach is suboptimal, as the training events order is likely to be broken.

For example, imagine the token sequence `[t, h, e, r, e]` on a certain training step. Tokens `(h, e)` are merged into `he` (event e_{i_1}). The sequence becomes `[t, he, r, e]`. Later, token `he` becomes useless and is removed (event $e_{i_2}, i_2 > i_1$). Thus, the sequence returns to `[t, h, e, r, e]`. It can happen now that tokens `(e, r)` are merged into a new token `er` (event $e_{i_3}, i_3 > i_2$). The resulting tokenization is `[t, h, er, e]`. PickyBPE tokenization will follow event order $e_{i_1}, ..., e_{i_2}, ..., e_{i_3}$ and result in `[t, h, er, e]`. The tokenization when the tokens are removed after the vanilla BPE process will first achieve `[t, he, r, e]`, as it will execute all the available merges. In a simplified example, there are no merges to perform after this step, and the algorithm will move to the removals phase: `he` will be split, and the resulting tokenization will become `[t, h, e, r, e]`. Therefore, `er` will not be merged, as it happened after the removal and contains a part of the removed token.

When repeated several times, the described issue may lead to undesired tokenization results and compromise compression. In Table 6, we compare the compression rates of the two methods. The compression issues become more pronounced with lower thresholds as more tokens are removed.

$\mathcal{T}$	BPE inference with post-removal	PickyBPE inference
1.0	1.000	1.000
0.9	0.998	**0.997**
0.8	0.998	**0.996**
0.7	1.000	**0.994**
0.6	1.005	**0.992**

Table 6: Comparison of compression rates (↓) for the vanilla BPE inference followed by splitting undesired tokens and PickyBPE inference by events order for EN–DE vocabularies of size 32768. The compression rates are shown for English.

Parameter	Value
Encoder layers	6
Decoder layers	6
Embedding dim	512
Hidden dim	1024
Attention heads	4
Max tokens in a batch	4096
Optimizer	Adam
Weight decay	1e-4
Learning rate (LR)	5e-4
LR Scheduler	inverse sqrt
Warmup steps	4000
Warmup strategy	linear
Precision	`fp16`

Table 7: `transformer-iwslt` architecture and training details configuration from `fairseq` (Ott et al., 2019).

Apart from the described inference methods, PickyBPE can use any inference method requiring a fixed vocabulary: for example, greedy left-to-right decoding (Wu et al., 2016) or recently introduced PathPiece (Schmidt et al., 2024).

B Training details

Table 7 shows the main model and training hyperparameters we used in every machine translation experiment. We trained models with small vocabularies of size 8192 for 20 epochs and the models with larger vocabularies for 25 epochs, each on a single NVIDIA A40 GPU (driver version 555.42.02, CUDA version 12.5).

C Experiments with Chinese

We separately applied PickyBPE to the Chinese data to test the algorithm on a language with a completely different writing system. We used the ET–ZH mixtures of parallel corpora assembled by Korotkova and Fishel (2024) for training and the development set of the FLORES benchmark (Goyal et al., 2022) for evaluation. Because there were over 11,000 unique characters in the Chinese dataset alone, the smallest vocabulary size we could use was 16384, and we used the vocabulary of size 16384 separately for input and output. The results for these experiments are presented in Table 8.

The Estonian–Chinese experiment was the only language pair to show no improvement in any benchmark for any threshold setting for PickyBPE compared to vanilla BPE. Crucially, however, there was no drop in performance, but rather performance was stable across all conditions. We argue this is due to inherent language-specific differences introduced by the orthography of Chinese, which meant that our experiment is not able to show any benefits of PickyBPE with Chinese. After adding all individual characters and letters for Chinese and Estonian, there were only a small number of vocabulary items available for the tokenizer to create with merges. PickyBPE mostly operates by removing less frequent merges, which would tend to occur further down in the merge list. In this experimental setting, the tokenizer does not learn merges that would otherwise be later in the merge list, because of the relatively small vocabulary size for Chinese.

Experiment	$\mathcal{T}$	BLEU (↑)	COMET (↑)
ET–ZH	1.0*	22.3 ± 0.9	0.428
	0.9	22.0 ± 0.8	0.428
	0.8	22.0 ± 0.9	0.427
	0.7	22.0 ± 0.9	0.428
	0.6	22.0 ± 0.8	0.426

Table 8: Machine translation results on FLORES-dev (Goyal et al., 2022) for the ET–ZH language pair. For every threshold $\mathcal{T}$, we report BLEU (Papineni et al., 2002) and COMET (Rei et al., 2020) scores on the translation task. For both metrics, the differences in scores were found statistically insignificant. $^*\mathcal{T} = 1.0$ represents the baseline vanilla BPE without intermediate token removal.

D Under-trained tokens inspection

Figure 5 shows examples of token embedding norm distributions for thresholds 0.6, 0.7, and 0.8. As we lower the threshold, the proportion of removed tokens gets larger. However, there is no change in their nature: we remove mostly infrequent tokens and add more frequent tokens with higher norms that are close to the overall distribution.

E Number of Added/Removed Tokens

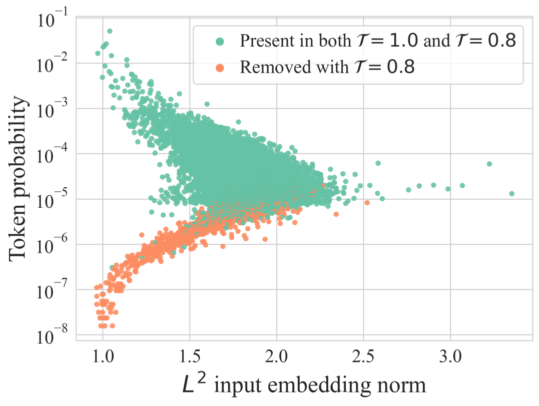
Tables 9, 10, and 11, report the number of added/removed tokens for each tokenizer. This is the number of tokens that are present in PickyBPE but not in the corresponding vanilla BPE tokenizer. With the growth of the vocabulary size, the overall partition of such tokens is maintained in a similar manner. It generally does not surpass 10% with the most radical threshold we use (0.6).

Vocabulary Size	Threshold	Added / Removed Tokens
8192	0.9	160
	0.8	358
	0.7	588
	0.6	805
16384	0.9	342
	0.8	707
	0.7	1092
	0.6	1468
32768	0.9	677
	0.8	1280
	0.7	1970
	0.6	2563
65536	0.9	1149
	0.8	2165
	0.7	3312
	0.6	4431

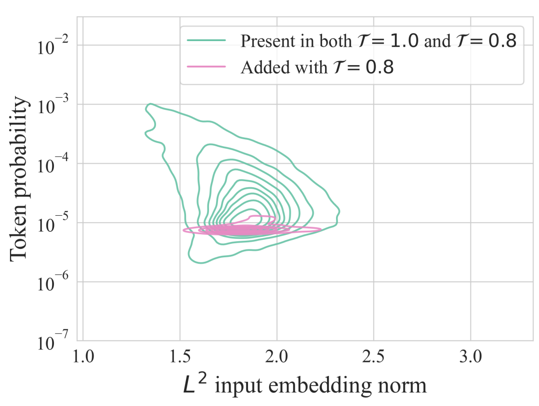
Table 9: Numbers of added (removed) tokens at different thresholds for the EN–DE tokenizers used for the translation experiments.

F Compression

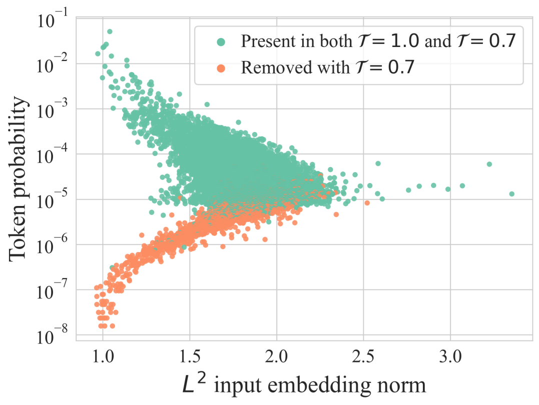
In Tables 12, 13, and 14, we show compression metrics as corpus token counts (CTC) for PickyBPE tokenizers relative to the vanilla BPE. We notice that compression is most pronounced in smaller vocabularies, as for the sizes of the datasets that we used larger vocabularies have large redundancy and a larger partition of tokens is allowed to be unused. In each experiment, we compute the CTC metric on the corresponding test sets.



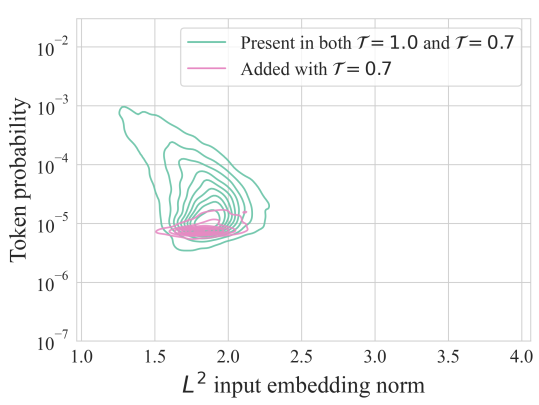
(a) PickyBPE tokens when $\mathcal{T} = 1.0$. The tokens that are present when $\mathcal{T} = 1.0$ but are removed when $\mathcal{T} = 0.8$ (orange) are generally infrequent and have low L^2 embedding norms, thus the majority of them are likely to be under-trained (Land and Bartolo, 2024).



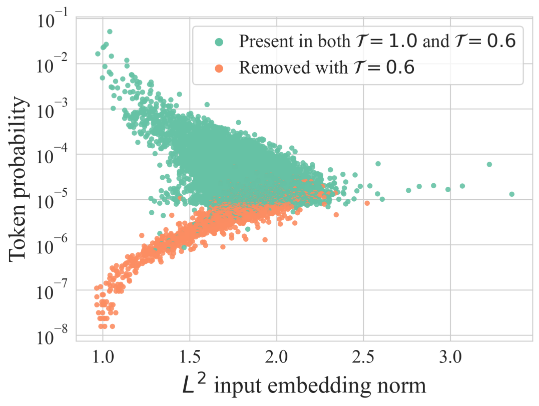
(b) PickyBPE tokens when $\mathcal{T} = 0.8$. The tokens that are present when $\mathcal{T} = 0.8$ but not when $\mathcal{T} = 1.0$ (pink) have frequencies and L^2 norms of the embeddings close to the blob center and thus are less likely to be under-trained (Land and Bartolo, 2024).



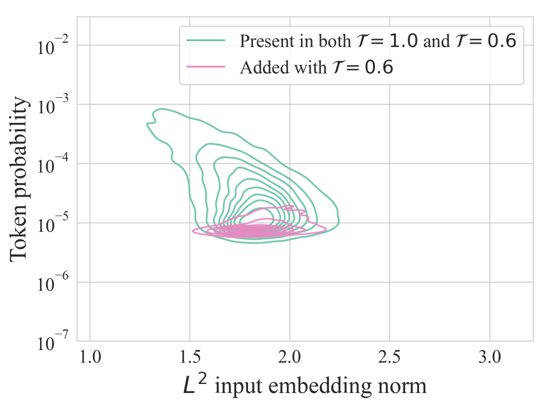
(c) PickyBPE tokens when $\mathcal{T} = 1.0$. The tokens that are present when $\mathcal{T} = 1.0$ but are removed when $\mathcal{T} = 0.7$ (orange) are generally infrequent and have low L^2 embedding norms, thus the majority of them are likely to be under-trained (Land and Bartolo, 2024).



(d) PickyBPE tokens when $\mathcal{T} = 0.7$. The tokens that are present when $\mathcal{T} = 0.7$ but not when $\mathcal{T} = 1.0$ (pink) have frequencies and L^2 norms of the embeddings close to the blob center and thus are less likely to be under-trained (Land and Bartolo, 2024).



(e) PickyBPE tokens when $\mathcal{T} = 1.0$. The tokens that are present when $\mathcal{T} = 1.0$ but are removed when $\mathcal{T} = 0.6$ (orange) are generally infrequent and have low L^2 embedding norms, thus the majority of them are likely to be under-trained (Land and Bartolo, 2024).



(f) PickyBPE tokens when $\mathcal{T} = 0.6$. The tokens that are present when $\mathcal{T} = 0.6$ but not when $\mathcal{T} = 1.0$ (pink) have frequencies and L^2 norms of the embeddings close to the blob center and thus are less likely to be under-trained (Land and Bartolo, 2024).

Figure 5: Input embedding vectors for PickyBPE tokens with **(a, c, e)** $\mathcal{T} = 1.0$, **(b)** $\mathcal{T} = 0.8$, **(d)** $\mathcal{T} = 0.7$, and **(f)** $\mathcal{T} = 0.6$ for English vocabularies of size 16384 in EN–DE experiments with separate vocabularies. For each token we compute its probability in the training corpus (y-axis), and the L^2 norm of its embedding vector in the trained model (x-axis).

Vocabulary Size	Threshold	Added / Removed Tokens
8192	0.9	133
	0.8	313
	0.7	506
	0.6	718

Table 10: Numbers of added (removed) tokens at different thresholds for the DE–ET tokenizers used for the translation experiments.

Vocabulary Size	Threshold	Added / Removed Tokens
8192	0.9	107
	0.8	255
	0.7	446
	0.6	605

Table 11: Numbers of added (removed) tokens at different thresholds for the UK–ET tokenizers used for the translation experiments.

Vocabulary size	$\mathcal{T}$	Compression (↓)	
		English	German
8192	1.0	1.000	1.000
	0.9	0.997	0.996
	0.8	0.995	0.993
	0.7	0.994	0.991
	0.6	0.992	0.989
16384	1.0	1.000	1.000
	0.9	0.996	0.998
	0.8	0.994	0.996
	0.7	0.993	0.995
	0.6	0.991	0.993
32768	1.0	1.000	1.000
	0.9	0.997	0.998
	0.8	0.996	0.998
	0.7	0.994	0.997
	0.6	0.992	0.996
65536	1.0	1.000	1.000
	0.9	0.998	0.998
	0.8	0.997	0.998
	0.7	0.997	0.998
	0.6	0.996	0.997

Table 12: Compression for EN–DE tokenizers with different vocabulary sizes reported as corpus token counts (CTC) relative to the vanilla BPE ($\mathcal{T} = 1.0$).

Vocabulary size	$\mathcal{T}$	Compression (↓)	
		German	Estonian
8192	1.0	1.000	1.000
	0.9	0.998	0.998
	0.8	0.994	0.996
	0.7	0.991	0.993
	0.6	0.989	0.991

Table 13: Compression for DE–ET tokenizers with a vocabulary size of 8192. The scores are computed as corpus token counts (CTC) relative to the vanilla BPE ($\mathcal{T} = 1.0$).

Vocabulary size	$\mathcal{T}$	Compression (↓)	
		Ukrainian	Estonian
8192	1.0	1.000	1.000
	0.9	0.998	0.998
	0.8	0.996	0.996
	0.7	0.993	0.994
	0.6	0.992	0.993

Table 14: Compression for UK–ET tokenizers with a vocabulary size of 8192. The scores are computed as corpus token counts (CTC) relative to the vanilla BPE ($\mathcal{T} = 1.0$).

G Word-Initial Tokens

In Tables 15, 16, and 17, we show the proportions of added and removed word-initial tokens for different vocabulary sizes and language pairs. By added tokens we mean the ones present in PickyBPE but not in vanilla BPE, and by removed tokens we mean the ones present in vanilla BPE but not in PickyBPE. In addition, we show the overall proportions of word-initial tokens in the corresponding tokenizers in Tables 18, 19, and 20.

H Token Length

In Tables 21, 22, and 23, we show mean token lengths over different vocabulary sizes that we used in the translation experiments. The token lengths are reported in characters.

Vocabulary size	$\mathcal{T}$	% Word-Initial Tokens	
		Dropped	Added
8192	0.9	43.8	65.5
	0.8	41.1	67.5
	0.7	42.0	66.9
	0.6	42.1	64.2
16384	0.9	43.9	69.6
	0.8	43.7	67.1
	0.7	45.3	68.1
	0.6	45.3	65.8
32768	0.9	46.7	73.3
	0.8	44.8	68.3
	0.7	47.5	68.5
	0.6	48.7	67.9
65536	0.9	50.6	74.6
	0.8	49.2	71.0
	0.7	51.5	69.9
	0.6	52.0	69.0

Table 15: Percent of word-initial tokens out of added and removed tokens for the EN–DE tokenizers. Added tokens are relative to the vanilla ($\mathcal{T} = 1.0$) tokenizer of the same vocabulary size and language pair.

Vocabulary size	$\mathcal{T}$	% Word-Initial Tokens	
		Dropped	Added
8192	0.9	33.1	60.9
	0.8	32.3	63.3
	0.7	37.0	60.3
	0.6	40.4	58.4

Table 16: Percent of word-initial tokens out of added and removed tokens for the DE–ET tokenizers. Added tokens are relative to the vanilla ($\mathcal{T} = 1.0$) tokenizer of the same vocabulary size and language pair.

Vocabulary size	$\mathcal{T}$	% Word-Initial Tokens	
		Dropped	Added
8192	0.9	31.8	73.6
	0.8	33.3	66.3
	0.7	37.4	61.8
	0.6	39.0	61.3

Table 17: Percent of word-initial tokens out of added and removed tokens for the UK–ET tokenizers. Added tokens are relative to the vanilla BPE ($\mathcal{T} = 1.0$) of the same vocabulary size and language pair.

Vocabulary Size	Threshold	% Word-Initial Tokens
8192	1.0	61.5
	0.9	61.9
	0.8	62.7
	0.7	63.3
	0.6	63.6
16384	1.0	68.0
	0.9	68.6
	0.8	69.2
	0.7	69.7
	0.6	70.0
32768	1.0	72.2
	0.9	72.8
	0.8	73.2
	0.7	73.6
	0.6	73.9
65536	1.0	75.2
	0.9	75.7
	0.8	76.1
	0.7	76.3
	0.6	76.6

Table 18: Overall proportion of word-initial tokens at different thresholds for the EN–DE tokenizers.

Vocabulary Size	Threshold	% Word-Initial Tokens
8192	1.0	58.1
	0.9	58.6
	0.8	59.4
	0.7	59.8
	0.6	60.0

Table 19: Proportion of word-initial tokens at different thresholds for the DE–ET tokenizers.

Vocabulary Size	Threshold	% Word-Initial Tokens
8192	1.0	59.8
	0.9	60.4
	0.8	60.9
	0.7	61.1
	0.6	61.5

Table 20: Proportion of word-initial tokens at different thresholds for the UK–ET tokenizers.

Vocabulary Size	Threshold	Mean Token Length (Chars) (↑)
8192	1.0	5.38
	0.9	5.40
	0.8	5.44
	0.7	5.47
	0.6	5.50
16384	1.0	6.19
	0.9	6.21
	0.8	6.24
	0.7	6.26
	0.6	6.28
32768	1.0	6.85
	0.9	6.88
	0.8	6.91
	0.7	6.94
	0.6	6.95
65536	1.0	7.44
	0.9	7.46
	0.8	7.49
	0.7	7.51
	0.6	7.53

Table 21: Mean token length at different thresholds for the EN–DE tokenizers.

Vocabulary Size	Threshold	Mean Token Length (Chars) (↑)
8192	1.0	5.35
	0.9	5.38
	0.8	5.40
	0.7	5.41
	0.6	5.42

Table 22: Mean token length at different thresholds for the DE–ET tokenizers.

Vocabulary Size	Threshold	Mean Token Length (Chars) (↑)
8192	1.0	4.84
	0.9	4.85
	0.8	4.86
	0.7	4.88
	0.6	4.90

Table 23: Mean token length at different thresholds for the UK–ET tokenizers.