FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jiří Balhar

Improving Subword Tokenization Methods for Multilingual Models

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	Ing. Tomasz Limisiewicz
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2023

I would like to thank Tomasz Limisiewicz for the time and effort he put into supervising this thesis. I am grateful for his guidance and insightful discussions, which not only shaped the outcome of this thesis but also, hopefully, refined my research skills.

I would like to thank Ondřej Dušek for all the friendly and encouraging calls we had before I started working on this thesis.

I am grateful to David Mareček for inspiring me to work on this topic and for connecting me with Tomasz.

Finally, I would like to thank my family and friends for their unwavering support and encouragement throughout my long and eventful studies. Special thanks go to my mother and brother for all the ”gofs”, to my father for sharing with me his endless curiosity about the world and science, and to my fiancée, who has always believed in me and supported me in every way possible.

Title: Improving Subword Tokenization Methods for Multilingual Models

Author: Jiří Balhar

Institute: Institute of Formal and Applied Linguistics

Supervisor: Ing. Tomasz Limisiewicz, Institute of Formal and Applied Linguistics

Abstract: In this thesis, we explore the differences between tokenization methods for multilingual neural language models and investigate their impact on language model representation quality.

We propose a set of metrics to evaluate the quality of tokenizations. We show that the metrics capture the differences between tokenizers and that they correlate with the downstream performance of multilingual language models.

Then, using our metrics, we assess why is the standard tokenizer training on a multilingual corpus reported to be ineffective for multilingual models. We investigate design choices such as data size, implementation or alphabet size. We identify that the issue might be caused by data imbalance and to solve it we propose to sample tokenizer training data uniformly.

We compare the standard tokenizer training with three proposed methods we replicate, that aim to mitigate the same reported issues. We show that the principle behind the improvements of the proposed methods is the same as with the uniform sampling.

Our findings offer a deeper understanding of tokenization methods for multilingual models. We propose a methodology and guidelines for training multilingual tokenizers. Lastly, we show how to achieve improvements in tokenization without the need for more complex tokenization methods.

Keywords: natural language processing|multilingual language models|subword tokenization|NLP

Contents

1 Introduction **3**
1.1 Contributions 5

2 Background **7**
2.1 Multilingual language models 7
2.2 Subword tokenization 9
2.2.1 Byte Pair Encoding (BPE) 9
2.2.2 Wordpiece 9
2.2.3 Unigram LM 10
2.3 Tokenization with many languages 12
2.3.1 Bias towards high-resource languages 13
2.4 Mitigating the language bias 14
2.4.1 Language-Clustered Vocabularies 14
2.4.2 Determining vocabulary capacity for each language . . 17
2.4.3 Combination of methods for scaling the vocabulary size 20
2.4.4 Other tokenization approaches 22

3 Methodology **23**
3.1 Data and languages 23
3.2 Vocabulary size 25
3.3 Data sampling 25
3.4 Tokenizer metrics 26
3.4.1 Characters per token 27
3.4.2 Average rank 28
3.4.3 Jensen-Shannon Divergence 32
3.4.4 Alphabet size and out-of-vocabulary tokens 33
3.5 Evaluation procedures 34
3.5.1 Types of experiments 34
3.5.2 Intrinsic evaluation 34
3.5.3 Extrinsic evaluation 35
3.6 Evaluation on downstream tasks 37

3.6.1 POS 37
3.6.2 NER 37
3.6.3 Dependency labeling 37
3.6.4 NLI 37
3.6.5 Sentence Retrieval 38
3.7 Implementation Details 38
3.7.1 Model pretraining 38
3.7.2 Model probing 38
3.7.3 Reproducing the vocabulary balancing methods 39

4 Tokenizer properties affect the performance of language models 45
4.1 Analysis of Tokenizer Properties 45
4.1.1 TokMix tokenizer 46
4.2 Results 46
4.2.1 Intrinsic evaluation 46
4.2.2 Extrinsic evaluation 47
4.3 Findings 52

5 Design choices for better multilingual tokenizers 53
5.1 Experiments 53
5.2 Results 54
5.2.1 Choice of implementation 54
5.2.2 Data size 55
5.2.3 Character coverage 55
5.3 Data imbalance 57
5.4 Findings 59

6 Vocabulary balancing methods 61
6.1 Experiments 61
6.2 Results 63
6.2.1 Comparison of balancing methods 63
6.2.2 Comparison of balancing methods per language 67
6.2.3 Comparison of balancing methods on downstream tasks 71
6.3 Findings 76

7 Conclusion 81
7.1 Limitations and future work 82

Bibliography 85

A Sentencepiece default parameters 93

Chapter 1

Introduction

Neural language models have been shown to perform well on a variety of Natural Language Processing (NLP) tasks and have become a de-facto standard for tackling tasks that require language understanding. The main advantage of these pretrained models is that they are able to leverage a large, unannotated pretraining corpus. After pretraining, we usually need only a fraction of annotated, task-specific data for finetuning the model for the task at hand. [1, 2] Multilingual pretrained models extend the pretrain-finetune paradigm to multiple languages. By pretraining on a large, *multilingual* corpus, these models provide high quality representations even for languages with low amount of training data ("low-resource languages"), where a dedicated monolingual model might not even exist. Furthermore, it has been shown that finetuning a multilingual model with data from a high-resource language makes the model capable to perform the task at hand in the other languages it has been pre-trained on. This "cross-lingual transfer", a phenomenon specific to multilingual models, allows the usage of the model for languages where task-specific data is not available. [3, 4]

Even though multilingual models are capable of narrowing the gap between high-resource languages and low-resource languages, they still suffer from the fact that the languages are not equally represented in the pretraining data. This leads to lower performance on the low-resource languages [5]. Furthermore, Conneau et al. [5] has shown that increasing the number of languages while keeping the model size fixed leads to a decrease in performance across all languages [5], which makes the representation of many languages hard.

Tokenization is the crucial first step we take when tackling an NLP problem. In simple terms, tokenization splits up an input text into a sequence of tokens — words or parts of words. The tokens are then used as the input for the NLP methods. Traditionally, the term tokenization referred to the methods of splitting up an input text into words as a preprocessing step for NLP methods. With the advent of large neural models, we have seen a shift towards using subword tokenization methods. The gist of these methods is that they do not stop segmenting at the word level but continue in splitting words into smaller subword units. This allows the tokenizers to represent words, that do not occur in training data as a sequence of more common subword units. Moreover, the vocabulary size of the tokenizers can be significantly reduced, compared to the word-level tokenization.

It has been shown that the choice of tokenization method has a significant effect on the performance of the NLP models - whether we talk about monolingual language models [6], multilingual models [7] or machine translation models [8, 9]. In the context of multilingual language models, the tokenization methods handle more than a hundred languages at once. This is a challenging task, as the languages differ in their morphology, writing system and other properties. Another challenge is, as mentioned before, the fact that the languages are not equally represented in the training data. Rust et al. [7] have shown that the low-resource languages are underrepresented in the tokenizer vocabularies. This leads to performance loss, which can be mitigated by changing the vocabulary to one with better representation [7].

The unsatisfactory performance of existing multilingual tokenizers on low-resource languages has led to the development of novel tokenization methods. Chung et al. [10], Zheng et al. [11], and Liang et al. [12] all propose methods that aim to improve multilingual vocabularies. Their methods are based on separating the training corpus into isolated subsets of monolingual corpora or corpora composed of similar languages only. On the subcorpora, they run the standard tokenization methods. After creating these language-specific vocabularies, they merge them back into a single vocabulary. The authors show that using their methods leads to an overall improved performance, especially for low-resource languages.

In this thesis, we ask what makes a good tokenization method for multilingual models, how to measure it and what factors influence it. Moreover, our aim is to achieve better tokenization for all languages, especially the ones that have been shown to be underrepresented in the previous multilingual models [7].

We fill a methodological gap by proposing a robust set of metrics for measuring the properties of tokenization methods in the multilingual setting. Our metrics measure whether tokenizers effectively represent meaningful, language-specific tokens in the vocabulary (*vocabulary allocation*) and whether the tokens they learn are shared across languages (*vocabulary overlap*). The questions we address are: **Q1:** How do subword tokenizers differ in overlap and allocation of learned vocabularies? And **Q2:** Which properties of multilingual tokenizers affect the language model representation quality?

After we establish our metrics, we address the underresearched question of

Q3: What is the reason that the standard tokenizer training method does not work well in the multilingual setting? To this end, we examine the effect of the training data size, character coverage, implementation choice and most importantly, the data imbalance between high-resource and low-resource languages present in the tokenizer training data. We find that the data imbalance has a significant effect on the resulting tokenizer and by balancing the training data, we can achieve better tokenization for the low-resource languages at an overall smaller cost on the high-resource ones.

Then we turn to the three existing works by Chung et al. [10], Zheng et al. [11], and Liang et al. [12], which we collectively refer to as ”vocabulary balancing methods”, proposed for improving tokenization for all training languages. We find that the three works report empirical improvements of their methods but compare themselves to highly unbalanced baselines. We, therefore, ask **Q4:** What is the effect of using the vocabulary balancing methods on the representation of low-resource languages? And **Q5:** How do the vocabulary balancing methods compare to the standard method of training the tokenizer on balanced and unbalanced joint corpus?

Through in-depth analysis, we find that the three methods we reproduce [10, 11, 12] improve tokenization by adjusting the *vocabulary allocation* between the languages. We further show, that in our setting, similar results can be achieved by a simpler method of uniform sampling of languages during the tokenizer training.

1.1 Contributions

The work on this thesis began as a collaboration with Ing. Tomasz Limisiewicz on the paper “Tokenization Impacts Multilingual Language Modeling”. The paper was accepted to the *ACL Findings 2023* and will be published in July 2023. In this thesis, we incorporate certain contributions from the paper, including:

- Proposal of metrics for measuring the quality of tokenization for multilingual models.
- Validation of the metrics on a variety of downstream tasks by comparing different tokenization methods.

The thesis continues in the multilingual tokenization research by:

- Establishing a comprehensive methodology for comparing multilingual tokenizers on the level of individual languages.
- Investigating the effect of important design choices in the tokenization training process on the resulting quality of the tokenization.

- Reproducing three tokenization methods proposed in the literature aimed at improving the representation of low-resource languages.
- Examining the working principle behind the reproduced methods and proposing a simpler alternative for achieving similar results as the more complex, replicated methods.

The thesis is structured as follows. In Chapter 2, we provide the necessary background on the multilingual models, tokenization methods and methods for analysis and improvement of multilingual tokenization. In Chapter 3, we propose our metrics, explain the evaluation schemes, define a common experimental setup and describe the implementation and replication details. In Chapter 4, Chapter 5 and Chapter 6, we present our experiments and findings. Finally, in Chapter 7, we summarize our results.

Chapter 2

Background

In this chapter, we describe our related work. We introduce multilingual language models and the tasks that are used to assess their performance. Then, we explain the concept of subword tokenization and the standard subword tokenization methods. Lastly, we show what problems have been identified when using standard tokenizers in multilingual settings and what methods have been proposed to mitigate these problems.

2.1 Multilingual language models

In recent years, we have seen a dramatic rise in the use of neural language models. The premise of these models is the ability to learn unsupervised from large amounts of unlabeled data.

We focus on studying the properties of **multilingual** language models following the works of Devlin et al. [1] and Conneau et al. [5], who introduced the models mBERT[1] and XLM-R, respectively. In these works, a Transformer [15] based multilingual masked language model is pretrained on sentences sampled from a multilingual corpus of around 100 languages. Training the models on a multilingual dataset allows the model to learn a shared representation for all languages. This shared representation can then be used for cross-lingual transfer, which the authors show on a variety of language understanding tasks.

The multilingual models such as mBERT or XLM-R are composed of the tokenizer, which splits the input text into tokens, the embedding matrix, which encodes the tokens into a sequence of word embeddings, and the Transformer encoder, which transforms the sequence of word embeddings into a sequence of

[1]We cite the monolingual BERT paper as the mBERT model does not have an associated publication. In the literature, sometimes the mBERT Github README is cited for the multilingual variant[14]

so-called contextualized embeddings. The contextualized embeddings are then fed into a task-specific output layer — usually a linear layer with a softmax activation. The pretraining is done using the masked language modeling task, where some of the input tokens are stochastically masked out and the model is trained to predict the masked tokens [1].

In the case of XLM-R [5], the pretraining is performed on data from 100 languages. The tokenization uses a joint vocabulary for all languages and the model is trained to predict the masked tokens in all languages at once. Even without any supervision, the model is shown to perform well on a variety of cross-lingual tasks.

The tasks that are usually used to evaluate the performance of the pretrained models are the following [16]:

- **Question Answering (QA):** This task involves predicting an answer to a question given a context. The context usually consists of a paragraph of text and the model must locate and return the correct answer from this text.

- **Natural Language Inference (NLI):** In this task, the model is given a pair of sentences and it has to determine the relationship between them: whether they contradict each other, they are logically consistent, or they are unrelated.

- **Part-of-Speech Tagging (POS):** This is a classic task in NLP where each word in a sentence is assigned a tag that represents its syntactic role (e.g., noun, verb, adjective, etc.).

- **Named Entity Recognition (NER):** NER involves identifying and classifying named entities in text into predefined categories such as person names, organizations or locations.

- **Cross-lingual Sentence Retrieval:** In this task, the model is given a query sentence in one language and a collection of candidate sentences in a different language. The goal is to retrieve the sentences from the candidate pool that are the translation of the query.

These tasks cover a broad spectrum of capabilities, testing basic linguistic understanding and complex reasoning skills. The performance of the pretrained model on these tasks provides a comprehensive measure of its cross-lingual abilities.

2.2 Subword tokenization

Subword tokenization methods split up words into subword units, which are then used as the input tokens for the neural models. This step is generally needed for pretrained models because including all words in the model vocabulary would be computationally infeasible. Subword tokenization allows scaling down the vocabulary size by splitting up rare words into subword units. Another advantage of subword tokenization over word tokenization is that it allows the model to generalize to unseen words by composing the embeddings of the subword units. For word tokenization methods, the missing words are called out-of-vocabulary tokens and subword tokenization methods mitigate largely this issue. In this section, we describe the most common subword tokenization methods used in large multilingual language models.

2.2.1 Byte Pair Encoding (BPE)

The Byte Pair Encoding (BPE) algorithm was introduced by Sennrich, Haddow, and Birch [17]. They have adapted a compression algorithm by Gage [18] to learn a subword vocabulary from a corpus. This method was shown to improve the performance of neural machine translation (NMT) models. BPE was subsequently used for some of the well-known pretrained models such as GPT-2 [2].

The principle of BPE is to iteratively merge the most frequent byte pairs in the corpus until the vocabulary size reaches the target size. The algorithm starts with a vocabulary of only the unique characters used in the input corpus. Using this initial vocabulary, we then compute the frequency of every character pair and merge the most frequent pair to create a new subword unit. We recompute the frequency of the subword unit pairs and repeat the process until the target vocabulary size is reached.

To tokenize an input text, the BPE algorithm then iteratively merges the pairs in the same order as they were merged during training. The algorithm stops when it has merged all pairs in the vocabulary. The output of the algorithm is a sequence of subword units that can be used as the input tokens for the neural model. We show the pseudocode of the BPE tokenizer training in Algorithm 1.

2.2.2 Wordpiece

First introduced in [19] the Wordpiece algorithm is a similar technique to BPE. It is used in the BERT [1] masked language model. The training is based on a greedy merging of the subword units similar to BPE. Unlike BPE, Wordpiece does not use the frequency of the subword units to merge them. Instead, it merges the

Algorithm 1 The Byte Pair Encoding algorithm.

```
function BPE($C$)
    $V \leftarrow$ unique characters in $C$
    while $|V| < V_{target}$ do
        $p \leftarrow$ most frequent pair in $V$
        $V \leftarrow V \cup \{p\}$
    end while
    return $V$
end function
```

pair that maximizes the likelihood of an unigram language model trained on the corpus.

Contrary to BPE and Unigram LM, there is no official, public implementation of the original Wordpiece algorithm. There are several implementations of the algorithm available but these diverge from the original algorithm. For example, the implementation in the Huggingface library [20] does not use the unigram language model to merge the subword units. Instead, the training follows the BPE procedure and only adds prefixes to the subword units to mimic the output format of the BERT Wordpiece implementation. [2]

The implementation in the Tensorflow library follows a different, top-down approach to creating the subword vocabulary. It starts with a vocabulary of words and then iteratively splits the words into subword units to reach a target vocabulary size. [3]

2.2.3 Unigram LM

The Unigram LM tokenizer, sometimes referred to as the Sentencepiece tokenizer after the name of the library that implements it [8], was introduced by Kudo [21]. The motivation for this method is to create a probabilistic model for subword tokenization. With this model, it is then possible to sample different segmentations of the input text. Training on these varied segmentations empirically improves the performance of the NMT models. In large language models such as XLM-R, the Unigram tokenizer is used deterministically, always choosing the most probable segmentation.

[2]We refer the reader to the Huggingface Wordpiece implementation `https://github.com/huggingface/tokenizers/blob/291b2e23ae81cf94738835852213ce120152d121/tokenizers/src/models/wordpiece/trainer.rs`

[3]We refer the reader to the Tensorflow documentation for the Wordpiece tokenizer `https://www.tensorflow.org/text/api_docs/python/text/WordpieceTokenizer`

Given an input sentence X, the Unigram LM algorithm finds the most probable segmentation $x^\star$ for the input sentence X:

$$x^\star = \arg\max_{x \in \mathcal{S}(X)} P(x) \tag{2.1}$$

Where $\mathcal{S}(X)$ is the set of all possible segmentations of the input sentence X given a subword vocabulary $\mathcal{V}$. The probability $P(x)$ of a segmentation x is computed as a product of subword occurrence probabilities $p(x_i)$:

$$P(x) = \prod_{i=1}^{|x|} p(x_i) \tag{2.2}$$

Here, the subword occurrence probabilities cannot be computed using occurrence statistics in the corpus since we do not have the final vocabulary yet. Also, with a given vocabulary, there are usually many possibilities on how to segment the input sentence from character level granularity to word level. Instead, the Unigram LM uses the Expectation-Maximization (EM) algorithm to estimate the marginal subword occurrence probabilities $p(x_i)$ over all segmentation variants. The EM algorithm maximizes the marginal likelihood $\mathcal{L}$ with respect to the latent subword probabilities $p(x_i)$:

$$\mathcal{L} = \sum_{s=1}^{|D|} \log(P(X^{(s)})) = \sum_{s=1}^{|D|} \log(\sum_{x \in \mathcal{S}(X^{(s)})} P(x)) \tag{2.3}$$

By maximizing the marginal likelihood, the Unigram LM considers all possible segmentations of the input sentences when estimating the subword occurrence probabilities $p(x_i)$. After optimization of $p(x)$, it is then possible to compute the most probable segmentation $x^\star$ for a given input sentence X using the Viterbi algorithm.

The training of the Unigram LM works in a top-down fashion. It starts with a large seed vocabulary of the most frequent substrings (character n-grams). The default setting is seeding the vocabulary with top 1 000 000 most frequent character n-grams with $n \leq 16$. These character n-grams are then iteratively pruned down to the target vocabulary size in the following way:

1. With a given vocabulary $\mathcal{V}$, estimate the subword occurrence probabilities $p(x_i)$ using the EM algorithm.

2. Segment the training corpus, sample the best segmentation for every sentence.

3. For each subword x_i in the vocabulary, compute the loss. The loss is defined as the decrease in the unigram language model likelihood if the subword is removed from the vocabulary.

4. Keep the top 80% of the subwords with the lowest loss.

5. Repeat this process with the new vocabulary $\mathscr{V}$ until the target vocabulary size is reached.

After the training, the Unigram LM algorithm outputs a subword vocabulary $\mathscr{V}$ and the corresponding marginal subword probabilities $p(x_i)$. Tokenization of a new input is then done by running the Viterbi algorithm to sample top-n segmentations of the input sentence.

2.3 Tokenization with many languages

Despite the recent advances in language modeling, the tokenization methods used in multilingual language models remain mostly unchanged. The first models trained on multiple languages, such as mBERT [1, 14] and XLM [22] use the same tokenization methods as their monolingual versions - namely the Wordpiece and BPE algorithms. Later language models such as XLM-R [5], mBART [23] or mT5 [24] use the Unigram LM algorithm to learn the subword vocabulary. The most recent multilingual generative models, such as the 176B parameter BLOOM model [25] or the XGLM [26] use the Byte-level BPE and Unigram LM algorithms respectively.

The methods these models use to tokenize the input text are the same as the ones used in the monolingual models. The main difference is that the tokenization training algorithm is run on all of pretraining data. This means that the tokenizer is trained on all languages at once.

As we have pointed out previously, the pretraining data is not evenly distributed across languages. High-resource languages such as English or Indonesian may have hundreds of times the amount of training data than low-resource languages such as Swahili or Amharic.[4]

To account for the discrepancy in the number of training examples per language, the training data is usually subsampled with a bias towards low-resource languages. This subsampling is performed both for the model pretraining and the tokenizer training [14, 22].

We describe the subsampling method in detail in section 3.3. In a nutshell, the subsampling is done by adjusting the probability $p(l)$ of sampling a line from a language l. The probabilities are exponentiated with a factor α and renormalized to sum to 1. With $\alpha = 0.0$, the subsampling is uniform across languages. With

[4]By high-resource and low-resource languages, we mean languages with generally a lot or little data available in the pretraining corpora such as Wikipedia dumps or internet crawls. This may not necessarily correspond to the number of speakers of the language.

$\alpha = 1.0$, the smoothing has no effect. The usual values chosen for this factor α are 0.7 [1]. 0.5 [22] and 0.3 [5]. Throughout this thesis, we will refer to this subsampling method and the factor α often. When we refer to the factor α, we mean the factor used for the tokenizer training. We will use a fixed, separate α for pretraining data sampling.

2.3.1 Bias towards high-resource languages

As Rust et al. [7] have shown, the bias towards high-resource languages however still persists, even after subsampling the training data. The authors compare the performance of the multilingual model mBERT and language-specific variants of BERT. By finetuning and evaluating on five different tasks across nine typologically different languages, they show that there is a performance gap between the multilingual and monolingual models. By further examination they determine that the performance gap may be explained by 1) pretraining data size and 2) the choice of tokenizer and its suitability for the tested language.

They show that the performance of mBERT can be improved for a specific language by switching to a monolingual vocabulary and retraining only the embedding layer of mBERT. The model with a dedicated vocabulary outperforms the vanilla mBERT on a variety of tasks, which indicates that the multilingual vocabulary of mBERT is not optimal.

But what changes when we switch to a monolingual vocabulary and why does it improve the multilingual model to the point it approaches the performance of the dedicated monolingual model? Rust et al. [7] propose to look at how often the tokenizer segments words. They show that for some languages, the multilingual tokenizer splits words into subwords drastically more often than the monolingual tokenizer. By changing the tokenizer, we improve the tokenization of the input text and thus the model performance.

For measuring this, Rust et al. [7] use a metric called *fertility* introduced by []. It is defined as the average number of subwords per word. They show that the fertility of the multilingual tokenizer is higher than the fertility of the monolingual tokenizer, especially for low-resource languages that are underrepresented in the training data. By measuring the correlation between the improvement in performance of the modified models and the improvement in fertility, they show that there is a statistically significant relationship between the two.

2.4 Mitigating the language bias

As we have described, Rust et al. [7] have firmly established that there is indeed a disparity between the performance of the multilingual model mBERT and similar monolingual models. They have shown that the disparity is in part caused by the tokenization method used in the multilingual model.

Other works have addressed this issue by proposing novel tokenization methods, that aim to improve the vocabulary of the multilingual model and increase its size to accommodate for all represented languages. The works of Chung et al. [10], Zheng et al. [11], and Liang et al. [12] all introduce new multilingual models that use an expanded vocabulary and a novel tokenization method for mitigating the language bias. Moreover, the authors all claim that the standard recipe of training a tokenizer on a concatenation of all languages in the training data is not optimal and that the performance of the model can be improved by using their method, especially for the low-resource languages.

In the following subsections, we will describe these three methods. First, we describe the method of Chung et al. [10], which replaces the standard method with a procedure of clustering similar language corpora together and training separate cluster-tokenizers before merging these tokenizers together. Then we describe the method introduced by Zheng et al. [11] which infers the optimal per-language vocabulary size and then trains separate, monolingual tokenizers for each language that are then merged. Lastly, we describe the method of Liang et al. [12], which combines the previous two methods and trains even larger vocabularies than the previous two methods.

2.4.1 Language-Clustered Vocabularies

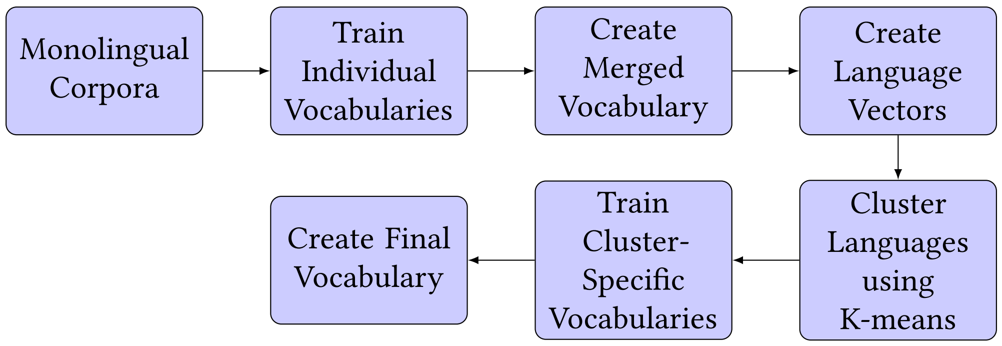

Monolingual Corpora
Train Individual Vocabularies
Create Merged Vocabulary
Create Language Vectors
Cluster Languages using K-means
Train Cluster-Specific Vocabularies
Create Final Vocabulary

Figure 2.1 Flowchart of the Chung et al. [10] method.

In their paper “Improving Multilingual Models with Language-Clustered Vocabularies” [10], Chung et al. propose a method to effectively increase the vocabulary size while mitigating the language bias by using language-clustered vocabularies. They construct an improved vocabulary by merging together several smaller vocabularies that were trained on subsets of the whole, multilingual training corpus. These smaller vocabularies are constructed by clustering the monolingual corpora based on their similarity and then training a separate vocabulary for each cluster. The authors show that the language-clustered vocabularies lead to improved performance on low-resource languages.

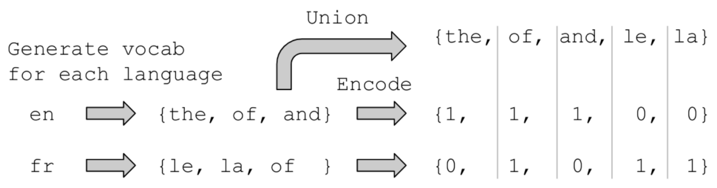


Figure 2.2 Binary vector representations used for language clustering. Figure taken from [10].

Their method works by clustering similar languages together and then merging the cluster-level vocabularies. To cluster the languages using the k-means algorithm, it is necessary to define an Euclidean vector space with each language having a representative vector (Figure 2.2). To this end, the authors first train equally sized vocabularies V^l for each language separately using the Unigram LM method. Then they create a merged vocabulary V^L by taking the union of the vocabularies V^l. Then, to produce a language vector $\mathbf{v}^l$ for each language l, the presence of each subword V_i^L is checked in the language-specific V^l and the value is set to 1 if the subword is present and 0 otherwise.

$$\mathbf{v}_i^l = \begin{cases} 1 & \text{if } V_i^L \in V^l \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

With the languages represented as vectors, the k-means algorithm can be used to cluster them. The authors use the cosine distance as the distance metric With 104 languages in total, the number of clusters is set to 8. The choice of k is determined empirically by running a set of preliminary experiments and comparing the results on a multilingual question-answering benchmark. The resulting clusters are shown in Table 2.1.

After the languages are clustered, the cluster-specific vocabularies are trained using the Unigram LM algorithm on the union of the corpora of the languages in the cluster. The size of each cluster-specific vocabulary is proportional to the size of the union of the individual (monolingual) vocabularies in the cluster. This means that more languages in a cluster lead to a larger vocabulary size assigned but also if the languages share tokens, this overlap decreases the vocabulary size. The final vocabulary is then constructed by simply taking the union over the cluster vocabularies.

The final vocabulary is then compared to the standard recipe of training a

Cluster	Languages	Vocab. size
c1	af, sq, hy, az, bn, bs, my, ceb, hr, en, fi, ka, el, he, hu, id, ga, jv, lv, lt, ms, min, pl, pa, sco, hbs, sl, su, sw, tl, th, tr, uz, vi, cy, yo	200,306
c2	ar, bpy, fa, azb, ur, lah	40,218
c3	an, ast, eu, ca, gl, io, it, la, lmo, oc, pms, pt, ro, scn, es, war	80,764
c4	ba, be, bg, ce, cv, kk, ky, mk, mn, ru, sr, tg, tt, uk	82,163
c5	bar, br, fr, de, ht, nds, lb, mg, vo	51,653
c6	zh-Hans, zh-Hant, ja, ko	25,528
c7	cs, da, nl, et, is, nb, nn, sk, sv, fy	57,681
c8	gu, hi, kn, ml, mr, ne, new, ta, te	61,683

Table 2.1 Clusters of languages used in [10]. The clusters are numbered from. Table taken from [10].

Unigram LM vocabulary on a joint corpus[5]. Because the proposed clustering method does not have a way of controlling the size of the final vocabulary as taking the union inadvertently leads to some tokens being merged, the authors compare the vocabularies by first following the clustering method and arriving at a 488k subword vocabulary. Then the standard method is followed to train a vocabulary of the same size.

The authors evaluate the vocabularies intrinsically - by examining the average number of tokens per sentence, out-of-vocabulary rate and vocabulary overlap using the Wasserstein-1 distance. By computing the average number of tokens per sentence for the whole corpus and for each language separately, the authors show that their method leads to a smaller average number of tokens per sentence for the whole corpus and further show that the improvements are more prominent for the low-resource languages. The authors also show that the out-of-vocabulary rate is lower for the language-clustered vocabulary, again the largest reductions in the OOV rate are in the low-resource and rare-script languages. Finally, the authors show on four selected languages that the language-clustered vocabulary leads to a smaller Wasserstein-1 distance between two similar languages and at the same time larger distance between two linguistically different languages[6].

The clustered vocabulary is then used to train a smaller and bigger multilingual masked language model and evaluated on three distinct, multilingual tasks

[5]The authors do not define what they mean by following the ”standard recipe”. Based on the related work the authors replicate closely (mBERT and XLM-R) and compare themselves to, we assume that the standard recipe is training a Unigram LM tokenizer on the pretraining data after the exponential subsampling described in 3.1. The authors do mention using the sampling factor $\alpha = 0.7$ for the pretraining data in the Appendix.

[6]We argue in subsection 3.4.3 that the Wasserstein-1 distance is not a suitable metric for measuring the distance between two vocabularies. It is a metric defined on probability measures and not on probability distributions.

(question answering, natural language inference and named entity recognition). The baselines selected are the original mBERT model, XLM-R model and a smaller reproduction of XLM-R with an increased vocabulary size to match the parameter size of the introduced model. The authors show improvements for the smaller model on all three tasks over their baseline reproduction. Then they show the improvements in QA and NER tasks for the bigger model over the XLM-R and mBERT baselines.

2.4.2 Determining vocabulary capacity for each language

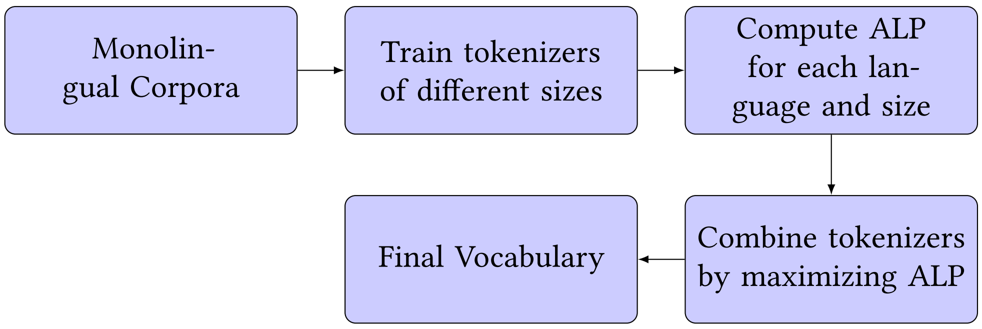

Monolingual Corpora
Train tokenizers of different sizes
Compute ALP for each language and size
Combine tokenizers by maximizing ALP
Final Vocabulary

Figure 2.3 Flowchart of the Zheng et al. method for determining vocabulary capacity.

In the paper “Allocating Large Vocabulary Capacity for Cross-Lingual Language Model Pre-Training”, Zheng et al. propose a method for determining the optimal vocabulary capacity for each language in a multilingual model. Further, they propose a method for constructing a multilingual vocabulary by combining monolingual vocabularies so that the optimal capacity is achieved for each language. In the second part of the paper, the authors propose a method for accelerating the training of the model with the increased vocabulary size, which is out of the scope of this thesis.

To determine the optimal vocabulary capacity for each language, the authors propose a metric called *average log probability* (ALP).

$$ALP(\mathscr{D}_i, V) = \frac{1}{|\mathscr{D}_i|} \sum_{j=1}^{|\mathscr{D}_i|} \sum_{k=1}^{|s_j|} \log(p_{uni}(s_j^k))$$

where $\mathscr{D}_i$ is the monolingual corpus of language i, V is the vocabulary and $p_{uni}(s_j^k)$ is the unigram probability of the k-th token in the j-th sentence. The authors show that the average log probability positively correlates with the downstream task performance on a series of experiments with four distinct languages. They show this correlation by training a series of tokenizers with increasing vocabulary size. Then they measure the ALP on these tokenizers, pretrain monolingual models for every vocabulary size and every language and evaluate them on two word-level classification tasks. For illustration, we show the correlation between ALP and F1 score for the NER task in Figure 2.4.

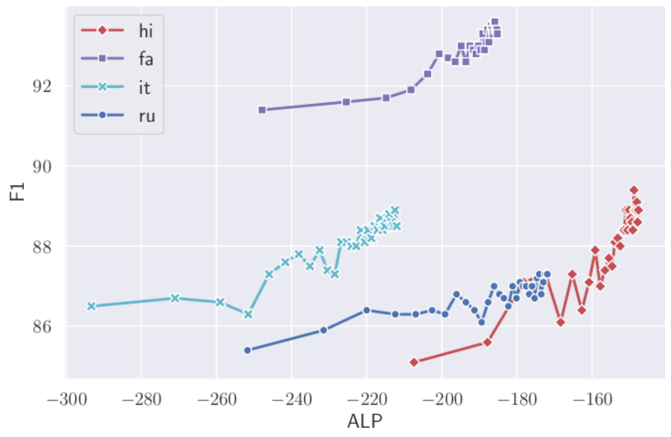


Figure 2.4 F1 score on NER task with different vocabularies versus their ALP on the monolingual corpus. Figure taken from [11]

Although the authors report that the vocabulary size itself is also a good predictor of downstream task performance, the authors argue that ALP correlates better with the downstream task performance and is therefore a better metric for determining the optimal vocabulary capacity.

To efficiently allocate the tokenizer vocabulary, the authors hypothesize that the pretraining size of the corpora must be also taken into account. The reason is that for low-resource languages, it is better to allocate fewer tokens as the language model does not have enough data to learn robust representations for the low-frequency tokens. In the end, the authors propose a greedy vocabulary allocation algorithm VoCap that maximizes the following objective:

$$\operatorname*{argmax}_{t_1,\ldots,t_N} \sum_{i=1}^{N} q_i^{\beta} \,\mathrm{ALP}\left(D_i, V_{t_i}^i\right) \text{ s.t. } \left|\bigcup_{i=1}^{N} V_{t_i}^i\right| = T$$

Where N is the number of languages, q_i is the probability of sampling training instances from i-th language during pre-training (we describe the per-language sampling in section 3.3), β is a hyperparameter that controls the importance of the corpus size and T is the total vocabulary capacity.

The VoCap algorithm works by precomputing a series of tokenizer vocabularies with vocabulary sizes from 1 000 to 50 000 for every language and computing the ALP metric for each of them. Then it iteratively builds the final vocabulary by:

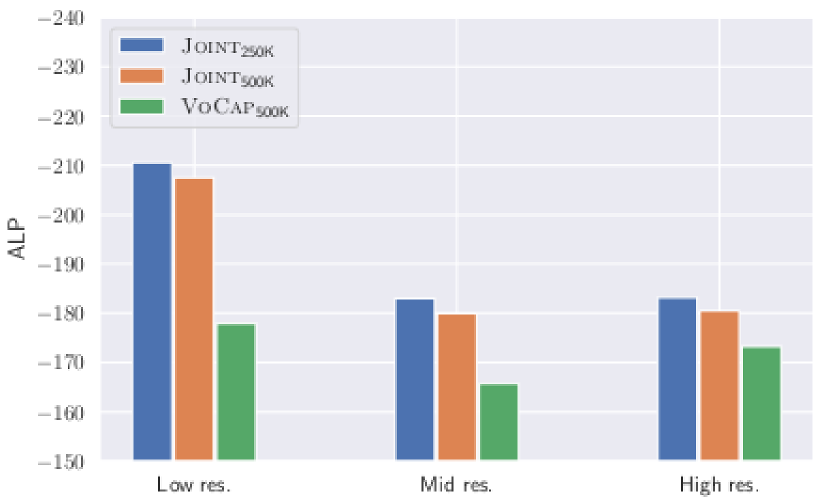


Figure 2.5 ALP improvement of VoCap tokenizers over the standard tokenizers with vocabulary sizes 250 000 and 500 000. Figure taken from [11].

1. Selecting the language with the highest potential ALP increase compared to the previous selected size.
2. Taking the tokenizer with the increased vocabulary size for the selected language.
3. Adding the tokenizer vocabulary to the final vocabulary.

After reaching the target size, the algorithm halts and returns the vocabulary constructed by the iterative merging.

With the VoCap algorithm, Zheng et al. then create new tokenizers with vocabulary sizes 250 000 and 500 000 on a multilingual corpus with 86 languages. The authors compare the performance of the VoCap tokenizers with baseline tokenizers trained directly on the multilingual corpus. The multilingual corpus is again subsampled before the standard tokenizer training following Devlin et al. [1] and Lample and Conneau [22] with an exponential smoothing factor of $\alpha = 0.7$.

The standard and VoCap tokenizers are compared using the ALP metric. The results are shown in Figure 2.5. The authors show that the VoCap tokenizers improve the ALP metric for the 500 000 vocabulary size over the baseline across all languages. The improvement is more prominent for low-resource and mid-resource languages compared to high-resource languages. At the same time the difference between the standard tokenizers with vocabulary sizes 250 000 and 500 000 is negligible.

To validate these results, the authors then pretrain masked language models using the standard and VoCap tokenizers and conduct experiments on tasks from the XTREME benchmark [27]. The comparison is done on natural language inference, paraphrase identification, part of speech tagging, named entity recognition and question answering. The averaged results over all tasks suggest that the VoCap tokenizers outperform the standard tokenizers by 0.4 percentage points for the 250 000 vocabulary size and 1.7 percentage points for the increased 500 000 vocabulary size. When investigating further the results for XNLI and NER, the authors show that improvements are gained in the low-resource and mid-resource languages. This is consistent with the ALP improvement results.

2.4.3 Combination of methods for scaling the vocabulary size

In the paper *XLM-V: Overcoming the Vocabulary Bottleneck in Multilingual Masked Language Models*, Liang et al. introduce a new multilingual language model with a 1M token vocabulary. To create this large vocabulary, the authors propose a new tokenization method by combining the two methods described in the previous sections 2.4.1 and 2.4.2.

The proposed method consists of the following steps: (1) training monolingual tokenizers for each language using Unigram LM, (2) computing language vectors using negative log probability of the tokens, (3) finding a clustering of the languages with the k-means algorithm using the language vectors, (4) finding an appropriate vocabulary size for each cluster using the ALP from Zheng et al. [11], (5) training a tokenizer for each cluster using Unigram LM and (6) combining the tokenizers into a single multilingual tokenizer.

As we can see, the overall method is similar to Chung et al. [10] with small adjustments. More concretely, the authors train larger monolingual Sentencepiece Unigram tokenizers in the first step, use a different method for computing the language vectors in the second step and use the ALP metric for finding the appropriate vocabulary sizes in the fourth step. We now describe each altered step in more detail.

For training the monolingual tokenizers, the authors suggest using a larger vocabulary size of 30 000 instead of 8 000 used in Chung et al. [10].

The language vectors in Chung et al. [10] are binary with 1 corresponding to each subword present in the vocabulary of a given language. In contrast, Liang et al. [12] propose to use the negative log probability of the subwords as the language vectors (see Figure 2.6). The logits are the output of the Unigram LM tokenization method as explained in subsection 2.2.3. The authors argue that "weighting each token by its likelihood better represents the lexical fingerprint of a language".

Lastly, the authors argue that the method of Chung et al. [10] assigns deficient

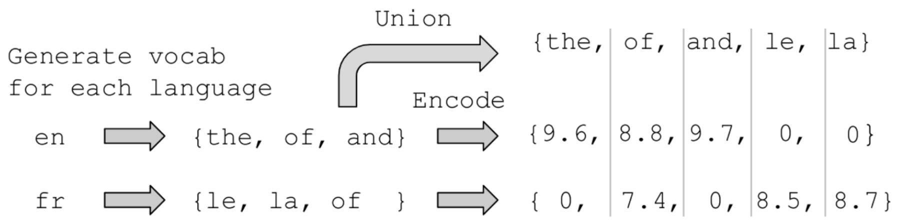


Figure 2.6 Negative log probability vector representations used for language clustering. Compare to Figure 2.2. Figure taken from [12].

vocabulary sizes to some of the clusters, pointing out an example of a cluster containing Chinese and Japanese with a capacity of 28,593 tokens. The authors propose to use the ALP to allocate appropriate vocabulary sizes to each cluster. The exact method is unclear from the paper but to the best of our understanding, the method followed by the authors was to use the publicly available code and monolingual tokenizers from Zheng et al. [11] and re-run the VoCap algorithm on their CC100 data instead of Wikipedia data. In this way, the authors obtained the optimized vocabulary sizes for the CC100 dataset for each language covered by Zheng et al. [11]. Any language that was not included in the public implementation of VoCap was set to a fixed size of 2000 vocabulary size. The optimized vocabulary sizes for each language were then summed according to the cluster assignments to get the cluster vocabulary sizes. To achieve a specified target vocabulary size (500k, 1M, 1.5M, 2M), the authors then take the cluster vocabulary sizes and rescale them so that the sum equals the target size. As noted in subsection 2.4.1, the union of the vocabularies is not guaranteed to be equal to the target size because of the overlapping tokens between the cluster vocabularies. The authors report that the 1M vocabulary has an actual size of 901 629 tokens.

After creating the vocabulary, the authors then train a large multilingual masked language model based on XLM-R following Conneau et al. [5].

They run a shorter preliminary training with differently sized vocabularies (1M and 1.5M) and compare the performance of their proposed method to the original clustering method by Chung et al. [10] scaled to 1M vocabulary size and XLM-R with the original tokenization method scaled to 1M vocabulary size. They compare the results on the XNLI task and show that their method outperforms the other two methods by 1.11 and 1.34 percentage points respectively.

After the preliminary run, they fully train the final model with the 1M vocabulary size on 2.5TB of data and compare the performance to the original XLM-R model with 250k vocabulary size. They show that their model significantly outperforms the original XLM-R model on a variety of language understanding tasks.

2.4.4 Other tokenization approaches

Several other works propose different methods for tokenization. In the thesis, we focus on methods that propose novel tokenization methods. There are however different approaches for text representation that aim to improve the performance of multilingual language models. We briefly describe some of them in this section.

Instead of costly pretraining of new models, Wang et al. [28] propose to extend the vocabulary of mBERT with new tokens. By adding new tokens, they lower the out-of-vocabulary rate for selected languages and in turn, improve performance on a variety of tasks.

One possible avenue for mitigating problems of tokenization is to get rid of tokenization altogether. Clark et al. [29], Tay et al. [30], and Xue et al. [31] propose different methods for skipping the input tokenization step and modifying the Transformer architecture to effectively process byte sequences. As Mielke et al. [32] notes, however, these methods come with their own sets of biases and limitations such as lower performance or higher computational demands.

Another direction of research is focusing on the visual representation of characters and subwords [33, 34, 35]. For example, logographic languages encode semantics in the shapes of the logograms, which is a source of additional information not present in the Unicode representation of the characters. The visual text representation models are also more robust to spelling errors and other artifacts.

Chapter 3

Methodology

In this chapter, we introduce the methodology for experiments conducted in this thesis. We define the data and languages used in our experiments. We also define a fixed vocabulary size to be comparable with related work. We introduce the important data sampling method utilized in Devlin et al. [1] and Conneau et al. [5] we use frequently for our experiments. Then we introduce our proposed metrics for measuring the *vocabulary allocation* and *vocabulary overlap* of a tokenizer and compare them to existing metrics used in the literature. We introduce the types of experiments we conduct — overall, we train different tokenizers and evaluate them against our metrics. Subsequently, we also use the tokenizers to train masked language models to verify that our metrics are useful for assessing the tokenizer quality for use in language models. We also describe in detail the evaluation procedures and all evaluation settings and tasks. Finally, we describe our reproduction of the vocabulary balancing methods by Chung et al. [10], Zheng et al. [11], and Liang et al. [12].

3.1 Data and languages

For training the vocabularies and the masked language models we follow our related work [5, 10, 12] and use the CC100 dataset. The other common choice is using Wikipedia data [1, 11] but the CC100 has been shown to improve low-resource languages such as Swahili and Urdu [5].

This unlabeled, multilingual dataset was created from the Common Crawl corpus using an automatic pipeline. The data was deduplicated and language-identified. Then for each monolingual corpus the data was filtered using Kneser-Ney language models trained on Wikipedia. Documents with perplexity under a certain language-specific threshold were filtered out. The data processing pipeline is described in detail in Wenzek et al. [36]. A reproduction of the dataset

Language	Language code	Script	Language Family	Data Size
English	en	Latin	Indo-European	214 431 709
Vietnamese	vi	Latin	Austroasiatic	99 914 417
Russian	ru	Cyrillic	Indo-European	80 356 837
French	fr	Latin	Indo-European	40 198 805
German	de	Latin	Indo-European	38 962 501
Spanish	es	Latin	Indo-European	37 190 928
Thai	th	Thai	Kra–Dai	31 773 751
Bulgarian	bg	Cyrillic	Indo-European	22 476 465
Hebrew	he	Hebrew	Afroasiatic	19 996 649
Chinese-simplified	zh-Hans	Chinese	Sino-Tibetan	19 655 905
Greek	el	Greek	Indo-European	19 773 458
Turkish	tr	Latin	Turkic	11 852 489
Arabic	ar	Arabic	Afroasiatic	11 421 836
Hindi	hi	Devanagari	Indo-European	10 326 155
Tamil	ta	Tamil	Dravidian	6 653 017
Georgian	ka	Georgian	Kartvelian	3 108 473
Urdu	ur	Arabic	Indo-European	2 669 302
Telugu	te	Telugu	Dravidian	1 716 206
Marathi	mr	Devanagari	Indo-European	1 206 981
Swahili	sw	Latin	Niger-Congo	1 232 106

Table 3.1 List of languages used in the experiments. Language sizes are reported in lines for the 10% subsample of the whole CC100 corpus.

is available at `https://data.statmt.org/cc-100/`.

For the purposes of this thesis, we select 20 out of 116 languages following Limisiewicz, Balhar, and Mareček [13] and download 10% of the data for each language. The reason for selecting a subset of the languages available and using only part of the data is our computational constraints. First, by limiting the amount of data, we can train the models faster. Second, by limiting the number of languages, we can scale down the vocabulary size. Because a large amount of the model parameters is in the embedding matrix, the vocabulary size has a large impact on the model size. This in turn, affects the training time and the memory requirements. We use the same diverse set of 20 languages for all experiments in this thesis.

The exact choice of the language subset is motivated by the downstream evaluation datasets. We use the 15 languages covered by XNLI and add 5 more. The languages are selected to cover a wide range of language families and scripts. The full list of languages is shown in Table 3.1.

3.2 Vocabulary size

We use a fixed vocabulary size of 120 000 tokens for representing the selected 20 languages. We have chosen this vocabulary size to create a comparable setup to our related work while saving our computational resources. The related work we replicate in this thesis uses the following vocabulary sizes:

- Chung et al. [10] use a vocabulary size of 488 000 tokens for 104 languages. This amounts to 4 692 tokens per language.
- Zheng et al. [11] use a vocabulary size of 500 000 tokens for 86 languages, which is 5 814 tokens per language.
- Liang et al. [12] use a vocabulary size of 901 629 tokens for 104 languages, which is 8 670 tokens per language.

The listed works propose to extend the vocabulary size over the previous models mBERT and XLM-R. For comparison, we also list the vocabulary size of these models:

- Conneau et al. [4] (XLM-R) use a vocabulary size of 250 000 tokens for 104 languages, which is 2 400 tokens per language.
- Devlin et al. [1] (mBERT) use a vocabulary size of 110 000 tokens for 104 languages, which is 1 058 tokens per language.

In our case, the vocabulary size of 120 000 tokens representing 20 languages amounts to 6 000 tokens per language. This is more than the vocabulary size of XLM-R and mBERT, but similar to the vocabulary size of the replicated works.

3.3 Data sampling

As explained in the Chapter 1 and shown in Table 3.1, the training data available for each language differs significantly in the total size (counted as the number of lines). For training the multilingual language model and associated tokenizer, it is generally advised to address this data imbalance by oversampling the languages with a low amount of data available (low-resource languages) and undersample the languages with high amounts of data (high-resource languages). One possible balancing procedure proposed by Devlin et al. [1] and Conneau et al. [5] is parametrized by the exponent α which we will now describe.

We assume we have N monolingual corpora C_l with languages $l \in L$. Each corpus with the language $l \in L$ has a different number of lines $N_l = |C_l|$. Then, the probability of sampling a line from the concatenation of all corpora $\cup_{l \in L} C_l$ is:

$$p(l) = \frac{N_l}{\sum_{l' \in L} N_{l'}} \tag{3.1}$$

To ensure that the low-resource languages are not underrepresented in the training data, we modify this probability distribution using an exponential smoothing parameter α:

$$p'(l) = \frac{p(l)^\alpha}{\sum_{l' \in L} p(l')^\alpha} \tag{3.2}$$

For $\alpha = 0.0$ we get a uniform distribution over the languages, for $\alpha = 1.0$ we get the original distribution.

For pretraining the language models, we use $\alpha = 0.3$ as suggested by Conneau et al. [4]. For training the tokenizers, we always specify the alpha as a parameter of the tokenizer training procedure.

3.4 Tokenizer metrics

In this section we introduce the metrics that we use to evaluate the tokenizers. By measuring the tokenizers we would like to explore two questions. First, we would like to analyse how the tokenizers differ between each other. How granular is the segmentation given an example text? And how much are the tokens shared between the languages? Second, using the observed differences between tokenizers, we would like to analyse how they influence the multilingual language models, that are trained using the tokenizers.

When assessing the multilingual tokenizers, we also want to focus not only on the overall properties but also investigate the quality of tokenization for the individual languages. This gives us a better understanding of the tokenizers and allows us to compare the tokenizers with each other given a target language. For this purpose, we will use monolingual evaluation corpora for each language. The metrics we define will be therefore functions of the tokenizer τ and the corpus C_l with the selected language l.

We introduce three metrics - average rank, characters per token and Jensen-Shannon divergence. The first two metrics aim to measure the "vocabulary allocation" of the tokenizer — the degree to which is the given language represented in the vocabulary. The third metric measures the "vocabulary overlap" between a given pair of languages — the degree of token sharing between two languages.

To define the metrics formally, we use the following notation [37]. Let Σ be a set of characters we call the alphabet. In our context, the alphabet is the set of all valid Unicode characters. We call a string $s \in \Sigma^*$ a line or equivalently a sentence. Finally, we call a multiset of lines $C_l = \{s_1, \dots, s_{N_l}\} \subset \Sigma^*$ a corpus of size N_l. The $l \in L$ denotes a language of the corpus from a set of languages L. Next, we denote the set $V_\tau \subset \Sigma^*$ as the vocabulary of a tokenizer τ. The tokenizer $\tau : \Sigma^* \to V_\tau^*$ is a mapping from a line $s \in \Sigma^*$ to a sequence of tokens $\tau(s) \in V_\tau^*$. We also denote $\tau(C_l) = \{\tau(s), s \in C_l\}$ the tokenization of the corpus C_l. We denote the length of a sequence of characters or tokens s as $|s|$. Finally, we denote the number of occurrences of a token $t \in V_\tau$ in a corpus C_l as $\text{cnt}(t, C_l)$ and the empirical probability of the token t in the corpus C_l as:

$$p(t, C_l) = \frac{\text{cnt}(t, C_l)}{\sum_{t' \in V_\tau} \text{cnt}(t', C_l)}$$

If the context is clear, we will omit the corpus C_l and write $\text{cnt}(t)$ and $p(t)$.

3.4.1 Characters per token

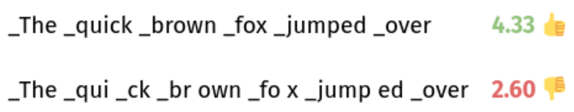

_The _quick _brown _fox _jumped _over
4.33
_The _qui _ck _br own _fo x _jump ed _over
2.60

Figure 3.1 Example of CPT metric. Figure from the paper poster [13]

The first metric we propose is the average number of characters per token (CPT). The motivation for this metric is that we want to measure how granular the tokenization for a given language is. If the tokenizer splits the words into many tokens, the average number of characters per token will be low. On the other hand, if the tokenizer does not split the words, the average number of characters per token will be high. We hypothesize, that longer tokens are better for the language models, because they potentially carry more meaning. The extreme case of this metric is the character-level tokenization, where the average number of characters per token is 1. In this case the model would need to learn to reconstruct the words from the characters.

The metric is defined as follows. Given a tokenizer τ and a language corpus C_l, we first tokenize the corpus using the tokenizer τ. Then we compute the average number of characters per token in the tokenized corpus:

$$CPT(\tau, C_l) = \frac{\sum_{s \in C_l} |s|}{\sum_{s \in C_l} |\tau(s)|} \tag{3.3}$$

The metric is illustrated in Figure 3.1.

The CPT metric is connected to the *average tokenized length* (or sequence length, or description length) metric used in Chung et al. [10] and Liang et al. [12]. These works suggest using the metric to compare whether one tokenizer splits a selected low-resource language into more tokens compared to another tokenizer. The average tokenized length is defined as the average number of tokens per sentence:

$$TL(\tau, C_l) = \frac{\sum_{s \in C_l} |\tau(s)|}{|C_l|} \tag{3.4}$$

The tokenized length can be expressed as the product of the reciprocal of CPT metric and a "average sentence length" constant, which is corpus-specific and not dependent on the tokenizer:

$$CPT(\tau, C_l)^{-1} \cdot \frac{\sum_{s \in C_l} |s|}{|C_l|} = \frac{\sum_{s \in C_l} |\tau(s)|}{\sum_{s \in C_l} |s|} \cdot \frac{\sum_{s \in C_l} |s|}{|C_l|} = \frac{\sum_{s \in C_l} |s|}{|C_l|} = TL(\tau, C_l) \tag{3.5}$$

Even though the metrics are equivalent, we use the CPT metric instead of the average tokenized length because we believe it is more intuitive (higher CPT is better) and it is easier to interpret thanks to the lower bound of 1 character per token.

CPT is also similar to another metric used in the literature — the word fertility metric used in Rust et al. [7]. The word fertility is defined as the average number of tokens per word. We can see that the same argument as in Equation 3.5 can be made about fertility and CPT. If we consider a corpus-specific constant "average number of characters per word", we see that fertility and CPT are proportional. The fertility metric has been shown to correlate with downstream performance and therefore seems to be a good metric. The downside of this metric is that it is not language-agnostic because it is not defined for languages without word delimiters such as Chinese or Thai.

3.4.2 Average rank

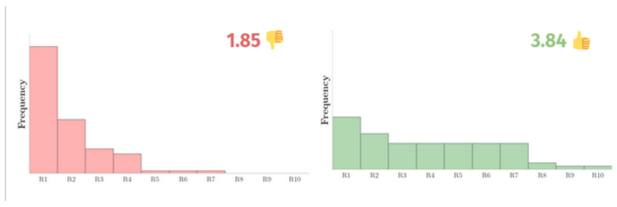


Figure 3.2 Example of AR metric. Figure from the paper poster [13]

Another metric we use for comparing the tokenizers is Average Rank (AR). The motivation for this metric is that we want to measure how many tokens are effectively used in the vocabulary for representing the corpus. Each language will have some amount of tokens dedicated to it in the vocabulary and our goal is to measure this allocation. We also want to take into account how frequently are these tokens used. We hypothesize that very frequent and very rare tokens are not as useful for the language models as the high-frequency tokens might be too ambiguous and low-frequency tokens might not have enough training examples to learn from [9]. We therefore propose to measure the average rank (the position of the token sorted by frequency) of tokens in the empirical probability distribution over a monolingual corpus.

Given a tokenizer τ and a language corpus C_l, we first tokenize the corpus using the tokenizer τ. Then we compute the empirical probability of the tokens in the tokenized corpus. We sort the tokens by their probability and assign them ranks from 1 to $|V_\tau|$. The average rank is then the weighted average of the ranks of the tokens, where the weights are the probabilities of the tokens:

$$AR(\tau, C_l) = \sum_{t \in V_\tau} rank(t, \tau(C_l)) \cdot p(t, C_l) \tag{3.6}$$

The metric is illustrated in Figure 3.2. Higher AR signals that the vocabulary contains higher number of tokens used for tokenizing given language. Moreover with high AR we can expect that the tokens are distributed more uniformly.

Average Rank and Average Log Probability

Now, we would like to address how our *average rank* compares to different metrics used in the literature.

First, we examine the *average log probability* (ALP) defined in Zheng et al. [11]. This metric is proposed for the same purpose as our AR metric. It is also said to measure language-specific vocabulary capacity and the authors claim that it is ”penalized by the subword units with low-frequency”. Surprisingly, we can show that the ALP metric is equivalent to the product of negative entropy and average tokenized length.

Given a monolingual corpus C_l and a tokenizer τ, the ALP is defined as [11]:

$$ALP(\tau, C_l) = \frac{1}{|C_l|} \sum_{s \in C_l} \sum_{t \in s} \log p(t) \tag{3.7}$$

We can simplify the original formula 3.7 by observing that we add up the log token probabilities $\log p(t)$ repeatedly by summing over all sentences and all tokens in sentences. Consequently, the sum can be simply expressed in terms of token occurrence multiplied by the log token probability. We denote V_τ the set of all tokens in the tokenizer vocabulary τ and $\mathrm{cnt}(t)$ the number of occurrences of token t in the corpus C_l. We can rewrite the ALP metric as follows:

$$ALP(\tau, C_l) = \frac{1}{|C_l|} \sum_{t \in V_\tau} \mathrm{cnt}(t) \log p(t) \tag{3.8}$$

From here we can, interestingly, derive a relationship between token length (Equation 3.4), information entropy and ALP metric as follows:

$$\begin{aligned}
ALP(\tau, C_l) &= \frac{1}{|C_l|} \sum_{t \in V_\tau} \mathrm{cnt}(t) \log p(t) & (3.9)\\
&= \frac{1}{|C_l|} \sum_{t \in V_\tau} \frac{\sum_{t' \in V_\tau} \mathrm{cnt}(t')}{\sum_{t' \in V_\tau} \mathrm{cnt}(t')} \mathrm{cnt}(t) \log p(t) & (3.10)\\
&= \frac{\sum_{t' \in V_\tau} \mathrm{cnt}(t')}{|C_l|} \sum_{t \in V_\tau} \frac{\mathrm{cnt}(t)}{\sum_{t' \in V_\tau} \mathrm{cnt}(t')} \log p(t) & (3.11)\\
&= \frac{\sum_{t' \in V_\tau} \mathrm{cnt}(t')}{|C_l|} \sum_{t \in V_\tau} p(t) \log p(t) & (3.12)\\
&= \frac{\sum_{s \in C_l} |\tau(s)|}{|C_l|} \sum_{t \in V_\tau} p(t) \log p(t) & (3.13)\\
&= TL(\tau, C_l) \cdot (-H(p)) & (3.14)
\end{aligned}$$

In step 3.13 we express the total number of tokens in corpus C_l by counting over all sentences $s \in C_l$ and summing the number of tokens in each sentence $|\tau(s)|$.

By our examination of ALP, it becomes evident that the metric is a composition of two already well-established metrics. Interestingly, these metrics are multiplied together, even though they seem to be inversely related. On the one hand, shorter tokenized sentence lengths are generally considered to be better (as a shorter tokenized sentence length means longer and more meaningful tokens), while on the other, a higher entropy is often deemed more desirable (a more uniform distribution is preferable to a more skewed one). One interpretation could be that high ALP is achieved when the vocabulary consists of a large number of short tokens that are similarly useful (have a uniform distribution). The authors, unfortunately, do not provide an analysis or discussion to shed light on this aspect.

In comparison to our average rank, ALP measures the number of used tokens and the uniformity of the distribution thanks to the entropy in the equation. On the other hand, the ALP metric is punished by the length of the tokens, which is counterintuitive. We therefore stick to our AR metric, which we deem more intuitive and does not suffer from this issue.

Average rank and entropy

Next, a natural question is what is the difference between average rank and information entropy. The entropy of the tokenized corpus is defined as follows:

$$H(\tau, C_l) = -\sum_{t \in V_\tau} p(t) \log p(t) \tag{3.15}$$

Recall that we define average rank as follows:

$$AR(\tau, C_l) = \sum_{t \in V_\tau} rank(t) \cdot p(t) \tag{3.16}$$

We see that entropy provides similar characteristics as AR in the sense that more uniform distributions result in higher entropy and more tokens dedicated to a language also result in higher entropy. As we can see, the formula for entropy and AR differ only in the sign and one of the multiplied terms. The sign is not important as we are only interested in the relative values of the metrics. The difference in the multiplied terms is that AR uses the rank of the token, while entropy uses the log probability of the token.

To proceed with the analysis, we will assume that the tokens follow Zipf's distribution. This is a reasonable assumption for natural language data. The Zipf's distribution is defined as follows:

$$p_{zipf}(t) = \frac{1}{rank(t) \cdot H_{|V_\tau|}} \tag{3.17}$$

Where $H_{|V_\tau|}$ is the $|V_\tau|$-th harmonic number used as a normalization constant. Taking the logarithm of $p_{zipf}(t)$, we get:

$$\log p_{zipf}(t) = -\log rank(t) - \log H_{|V_\tau|} \tag{3.18}$$

Now if we plug in the log probability of the token into the entropy formula and leave out the constant as we are interested only in the relative values with a fixed vocabulary size, we get:

$$H(\tau, C_l) \propto \sum_{t \in V_\tau} p(t) \log rank(t) \tag{3.19}$$

We see that under the Zipfian assumption, the entropy may be viewed as an "average log rank". We empirically check this by computing average rank, average log rank, and entropy the Figure 3.3.

This provides us with an intuitive understanding of the difference between the two metrics. AR and entropy can be viewed as being related, with the difference being in their sensitivity to the rank of the tokens. AR, being directly related to

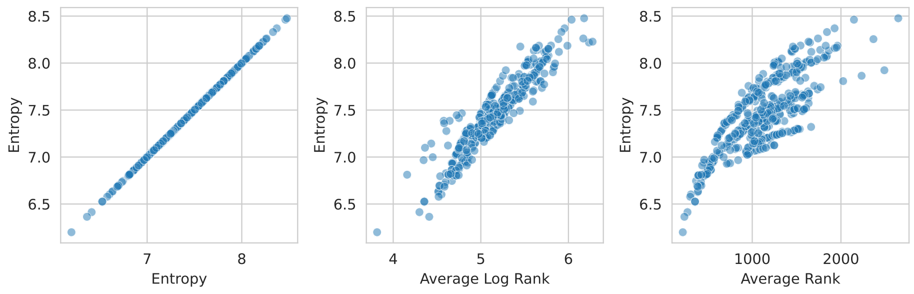
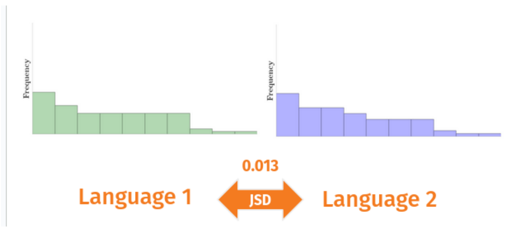


Figure 3.3 We compute the average rank, average log rank, and entropy for different tokenizers and languages. Then we plot each metric against entropy. We see that entropy and average log rank are similar, which supports our assumption that the tokens follow Zipf's distribution.

rank, is more sensitive to changes in probability in lower-frequency tokens. This is because the weighted average used in AR is more affected by linear rank values than the logarithmic rank values used in entropy calculation.

3.4.3 Jensen-Shannon Divergence



.

Figure 3.4 Jensen-Shannon Divergence. Figure from the paper poster [13]

The Jensen-Shannon Divergence (JSD) is a metric that measures the similarity between two probability distributions. It is defined as follows:

$$JSD(p,q) = \frac{1}{2} \cdot (KL(p||m) + KL(q||m)) \tag{3.20}$$

Where $m = \frac{1}{2} \cdot (p+q)$ is the midpoint distribution and $KL(p||q)$ is the Kullback-Leibler divergence.

$$KL(p||q) = \sum_{t \in V_\tau} p(t) \log \frac{p(t)}{q(t)} \tag{3.21}$$

We use JSD for the analysis of an overlap between two languages given a tokenizer. Tokenization of two monolingual corpora C_{l_1} and C_{l_2} with the same tokenizer τ results in two probability distributions over the vocabulary V_τ. To compute the dissimilarity between the languages, we compute the JSD between those distributions $JSD(p_{\tau(C_{l_1})}, p_{\tau(C_{l_2})})$.

The JSD is a symmetric metric that is bounded between 0 and 1. Low JSD means that the two distributions are similar, high JSD means that the two distributions are different. We say that there is a high vocabulary overlap between two languages if the JSD is low and vice versa. The metric is illustrated in Figure 3.4.

Overlap in tokenization has been studied in Wu and Dredze [38], although the metric used there was the absolute number of overlapping tokens. The benefit of using Jensen-Shannon divergence for measuring the vocabulary overlap is that the metric takes into account the occurrence of the shared tokens. This is important because some of the overlapping tokens may be for example infrequent emojis or other special tokens that do not carry much information about the language nor the actual overlap in more meaningful tokens.

Chung et al. [10] uses the Wasserstein distance (or the "earth mover's distance") to measure the overlap between two languages. We believe this is not a suitable metric as the Wasserstein distance is defined for probability measures (probability distributions on a given metric space). The tokenizer vocabulary has no metric structure and the authors of the article do not specify how they define the metric on the vocabulary. It is therefore unsuitable to use the Wasserstein distance for measuring the overlap between two tokenizers.

3.4.4 Alphabet size and out-of-vocabulary tokens

Among the metrics we propose, we also measure the alphabet size of the tokenizers defined as the number of tokens of length 1 in the vocabulary:

$$\text{Alphabet} = |\{t \in V_\tau, |t| = 1\}| \tag{3.22}$$

We also measure the number of out-of-vocabulary (OOV) tokens in the corpus. The UNK token is a special token that used to represent all the tokens that are not present in the vocabulary V_τ. We measure the number of UNK tokens in the corpus as follows:

$$\text{OOV} = \sum_{t \in \tau(C_l)} \mathbb{1}_{t=\text{<UNK>}} \tag{3.23}$$

Where $\mathbb{1}_{t=\text{<UNK>}}$ is an indicator function that is 1 if the token is UNK and 0 otherwise. Because we use the same validation set for all the tokenizers, we can compare the number of OOV tokens directly between the tokenizers. Note that in the literature, the out-of-vocabulary tokens are measured as a OOV rate (number of OOV tokens divided by the total number of tokens in the corpus). We do not use this metric because the values of the OOV rate are small for the subword tokenizers and therefore slightly harder to compare. We acknowledge that this choice is specific to our setting where the validation set is the same for all the tokenizers. In the general case, the OOV rate is a better metric.

3.5 Evaluation procedures

In this section, we introduce the evaluation procedures that we use to evaluate the tokenizers. We first describe the types of experiments we do with the tokenizers. Then we explain the intrinsic evaluation procedure. Then we describe the extrinsic evaluation procedure.

3.5.1 Types of experiments

Generally, we can distinguish between two types of experiments we do with the tokenizers we compare in this thesis. The first type of experiment is comparing different tokenizers to each other using the evaluation metrics we introduced in the previous section. For example, we can compare the Unigram LM tokenizer to the BPE tokenizer. To do this, we will train the tokenizers on the same training corpus and then evaluate them using the intrinsic evaluation metrics on validation sets for all the languages L.

The second type of experiment is comparing different tokenizers on downstream tasks. For example, we can compare the Unigram LM tokenizer to the BPE tokenizer on the task of natural language understanding. To do this, we will train the tokenizers on the same training corpus and then use the tokenizers to train otherwise identical language models. Then we will evaluate the language models on test sets for all the languages L.

3.5.2 Intrinsic evaluation

For the intrinsic evaluation of a set of tokenizers, we compute the metrics we introduced in the previous section on the validation sets from all the languages L. We compute the metrics for each tokenizer and validation language separately. Then, we compare the per language metrics between the tokenizers. Because the metrics are generally different for different languages (for example the characters per token for Chinese will be always smaller than the characters per token for English), we will often compare the relative differences between the tokenizers rather than the absolute values.

We will also compute the overall metrics for the whole set of languages L. We do this by averaging the metrics over all the languages. We use the macro average over languages, which means that we average the metrics for each language with the same weight. We use the macro average because we want to assess equally the impact on the high-resource and low-resource languages. This is a choice we make to assess each language equally. Different weighting schemes might be more appropriate for different applications. Equal weighting across languages is also in line with how are multilingual models evaluated in the related work [16].

The JSD metric which measures the *vocabulary overlap* is defined between a pair of languages, measuring the overlap between the two. We compute the overall overlap by considering the JSD values for all pairs of languages: $l_1, l_2 \in L, l_1 \neq l_2$.

We can also define a "combined CPT" or "combined AR" for a pair of languages by computing the metric for both languages and then averaging the values.

3.5.3 Extrinsic evaluation

For the extrinsic evaluation of tokenizers, we compare the performance of corresponding language models that differ only in the tokenizers used. We evaluate the performance on a set of language understanding tasks. Because the language models are identical except for the tokenizer used, the differences in the performance will be caused only by the differences in the tokenizers and random factors such as the weight initialization of the language models.

Concretely, given a set of tokenizers, we train corresponding masked language models with the same architecture and pretraining data. We then evaluate the models by training linear classifiers (probes) on top of the contextualized word embeddings produced by the language models. For a given pretrained model, we train probes for each task-language combination utilizing training data in all languages given for the task. For each configuration, we train 3 random initializations of the probe with different seeds to acquire more stable results and an estimate of the variation. In total, $N_{\text{models}} \times N_{\text{tasks}} \times N_{\text{languages}} \times N_{\text{seeds}}$ probes are trained to compare the performance of the models for one experiment.

We use probing [39, 40, 41] to evaluate the language modeling capability of the models. Instead of finetuning we freeze the base model and train only linear classifiers on top of the outputs of the base model. This approach allows us to evaluate the language modeling capability of the models without the influence of the finetuning procedure.

To evaluate a model m on task t and languages t_L, we will train probes $f_{m,t,l_{\text{src}}}$ for each training (source) language $l_{\text{src}} \in t_L$. Then the probe will be evaluated on the task test sets in languages $l_{\text{tgt}} \in t_L$ using standard classification metric (in our case: accuracy or F1 score).

Evaluation schemes

We will distinguish between two evaluation schemes — in-language evaluation and cross-language evaluation.

The in-language performance of the model m for task t and a language l will be computed by evaluating the probe $f_{m,t,l}$ on the test set for the same language l. The overall in-language performance of the model m for task t will be computed by averaging the in-language performance over all the languages $l \in t_L$.

The cross-language performance of the model for task t from the source language l_{src} to the target language l_{tgt} will be computed by evaluating the probe $f_{m,t,l_{\text{src}}}$ on the test set for a different language $l_{\text{tgt}} \neq l_{\text{src}}$. The overall cross-language performance of the model m for task t will be computed by averaging the cross-language performance over all the language pairs $l_{\text{src}}, l_{\text{tgt}} \in t_L, l_{\text{src}} \neq l_{\text{tgt}}$. Moreover, we will compute the cross-language performance per language l of the model m for task t by averaging the cross-language performance over all the languages $l_{\text{src}} \in t_L$ given a target language l.

When considering the results per language in both evaluation schemes, it is useful to consider only the relative differences between the models rather than absolute values. The performance of the models for a given language is influenced by eg. by the amount of training data available for the language. Therefore, when interpreting the results per language, we will choose a reference model and compute the relative difference between the performance of the reference model and the other models. Alternatively, we compute the mean performance for a given language across the models and report the relative difference between the models and the mean performance.

Correlation between intrinsic and extrinsic evaluation

To support the claim that intrinsic evaluation is a good proxy for extrinsic evaluation, we will compute the correlation between intrinsic and extrinsic evaluation. Because the tokenizer metrics and model performance are influenced by the evaluation language as mentioned in the previous paragraph and in subsection 3.5.2, we center the tokenizer metrics and downstream task results by subtracting the mean for each language in the in-language setting or pair of languages in the cross-lingual setting. In both cases, means are computed across all tokenizers. We present Spearman's correlation coefficient and the associated p-value.

Variation estimation

To account for the inherent randomness of the training procedure, we will train multiple probes for each configuration (m, t, l). We will use 3 random seeds for each probe and report the average performance over the seeds. We will also report the standard deviation of the performance over the seeds. In the case of the summarized performances, we estimate the standard deviation using bootstrapping over the seeds.

Note that because for one tokenizer we pretrain only one model, as it is the most costly part of the experiment, we do not estimate the variance for the pretraining of the model.

3.6 Evaluation on downstream tasks

Here we present the downstream tasks we use in our paper [13]. For our further experiments, we use a subset of these tasks (POS, NER, NLI) that we have found to have different responses to the changes in the CPT, AR and JSD metrics.

3.6.1 POS

We use Part of Speech annotations from Universal Dependencies [42]. The dataset is available for 17 languages analyzed by us (not covered: Swahili, Thai, Georgian). Each word is assigned one of the 17 coarse POS tags. We report the F1 score.

3.6.2 NER

We use the Wikiann dataset [43] consisting of Wikipedia articles with annotated named entities of three types: location, person, and organization in IOB2. Following XTREME, we use balanced data splits from [44]. We report the F1 score using the `seqeval` library.

3.6.3 Dependency labeling

As in Part of Speech, we use Universal Dependencies [42] for the dependency relation annotations. We use the largest UD treebank available for each language. For each word, we predict one of the 37 universal relations to its head word.

3.6.4 NLI

We use XNLI dataset [45] for Natural Language Inference. We classify whether the premise contradicts, entails, or is neutral to the hypothesis. We evaluate XNLI with the accuracy of classification.

XNLI contains data for 15 languages (not covered: te, ta, mr, he, ka).

3.6.5 Sentence Retrieval

We use up to 1,000 sentences aligned for pairs of languages from Tatoeba dataset [46]. For the pairs including English, we use the same sample as in XTREME data collection. For other pairs, we perform sampling ourselves.

We compute the cosine similarity between sentence representations across languages and find the best alignment with the Hungarian algorithm[47]. We compute the accuracy as the number of correctly aligned sentences divided by the total number of sentences.

3.7 Implementation Details

3.7.1 Model pretraining

We will train the language models to be able to assess the influence of the tokenization method on the language model performance.

We use the Huggingface framework [20] for pretraining the language model. We follow Conneau et al. [5] and pretrain the language model with the Masked Language Model objective. The model architecture is based on a scaled-down version of XLM-R_{Base} [5]. The size of embeddings is kept at 768, the number of attention layers is reduced from 12 to 8, the number of attention heads is reduced from 12 to 6. The maximum sequence length is set to 128 tokens. The total number of parameters is roughly two times smaller than the original XLM-R_{Base}.

The models are pretrained for 10k steps with the batch size 8192 achieved by using gradient accumulation. This amounts to ≈ 1.6 epochs over our training dataset. The learning scheduler is linear with warmup for the first 500 steps. The learning rate is set to $5e-4$. We use the AdamW optimizer.

3.7.2 Model probing

To evaluate the models, we train a linear classifier on top of the model. For the word-level classification tasks (POS, NER), the probe is trained on the mean of the word embeddings. For dependency labeling, the probe is trained on the concatenation of the two word representations along with their element-wise product as an input to the probe ($[h_{w1}; h_{w2}; h_{w1} \odot h_{w2}]$). For sentence-level XNLI, we train the linear classification probe on top of the concatenation of two sentence vectors and their element-wise product: $[h_{s1}; h_{s2}; h_{s1} \odot h_{s2}]$.

We freeze the model parameters and train only the classifier. We use the Adam optimizer with a learning rate $2e-3$ and batch size 512. We train for 60 epochs and select the best model based on the validation accuracy.

The code used for pretraining and finetuning is included in the thesis attachment.

3.7.3 Reproducing the vocabulary balancing methods

In this section, we describe our reproduction of the existing methods for balancing the low- and high- resource languages in the vocabulary. As the code for two out of three of the methods is not available, we follow and reimplement the original papers closely and describe the differences in our implementation.

The methods of Chung et al. [10] and Liang et al. [12] follow a three step process: 1) grouping the languages into clusters by similarity, 2) running the Unigram LM tokenizer training on the clustered corpora and 3) combining the cluster-vocabularies into a single, multilingual vocabulary. Because of their similarity, we refer to the two as the clustering methods. The method of Zheng et al. [11] works in two steps: 1) training the Unigram LM tokenizers for each language separately and 2) selecting the best vocabulary size for each language and combining the vocabularies into a single, multilingual vocabulary.

As we can see the methods share the last merging step and differ in the clustering approaches. We therefore describe the clustering approaches first, then we describe the Zheng method and finally we describe the merging step common for all methods.

Reproducing the clustering methods

The first step for the Chung and Liang methods is to train monolingual Unigram LM tokenizers for each language l from the set of 20 languages L. As specified in Chung et al. [10], we use the default Sentencepiece settings we describe in Table A.1. For training the monolingual tokenizers, we use 1M lines for each language which we have shown to be enough in preliminary experiments subsection 5.2.2. This choice also corresponds to the Zheng et al. [11] method, which uses 1M lines for each language. The vocabulary size differs between Chung and Liang and so we train two sets of monolingual tokenizers, with 8k and 30k vocabulary sizes respectively.

We arrive at $|L|$ vocabularies V^l. Next, we take the union of all vocabularies $V^L = \bigcup_{l \in L} V^l$ and compute the "vector representation" $\mathbf{v}^l$ for each language l as described in subsection 2.4.1 and subsection 2.4.3. For Chung, we compute a binary vector of size $|V^L|$, where each element $\mathbf{v}^l_i$ is 1 if the token i is in the vocabulary V^l of language l and 0 otherwise. For Liang, we compute a vector of size $|V^L|$, where each element $\mathbf{v}^l_i$ is the negative log-probability of the token i in the language l as computed by the Sentencepiece training algorithm. We set the log probability of the tokens not in the vocabulary to 0 as inferred from the Figure 2.6 from [12][1].

With the vector representations $\mathbf{v}^l$ we can cluster the languages into k clusters C^k using the k-means algorithm. We use the implementation from `scikit-learn` [48] with the default parameters. Chung et al. [10] reports using the cosine distance for the k-means algorithm. We normalize the language representation vectors to unit length, to achieve the same effect. In the case of Liang, we stick to Euclidean distance as the authors do not mention using a different metric. We experiment with $k \in \{4, 8, 16, 20\}$. Note that $k = 20$ corresponds to separating each language into a separate cluster which is similar to the method TokMix we introduce in Limisiewicz, Balhar, and Mareček [13] and Zheng et al. [11] we replicate next.

Then, given a clustering of languages C, for each cluster $c_j \in C$ we create a new training corpus by concatenating all of the CC100 data belonging to the cluster[2]. We run the Sentencepiece algorithm again on these clustered corpora to arrive at cluster-specific vocabularies V^{c_j}. The vocabulary size for each cluster is determined following the Chung and Liang methods. For both methods, we want to arrive at the final size of 120k tokens after merging the cluster-specific vocabularies. Therefore we need to determine the size of each cluster vocabulary $|V^{c_j}|$ such that $\sum_{j=1}^{k} |V^{c_j}| = 120k$. The Chung method sets the size of the cluster vocabulary to be proportional to the size of the union over the monolingual vocabularies $|\bigcup_{l \in c_j} V^l|$ to determine the size of each cluster vocabulary as follows:

$$|V^{c_j}| = \frac{|\bigcup_{l \in c_j} V^l|}{\sum_{i=1}^{k} |\bigcup_{l' \in c_i} V^{l'}|} \cdot 120k \tag{3.24}$$

The Liang method proposes to set the size of the cluster vocabulary to be proportional to the sum of the vocabulary allocations from Zheng et al. [11] for the languages belonging to the cluster. We use the allocations we reproduce in section 3.7.3. [3]

[1]We note that setting the log probability of tokens not present in vocabulary to 0 might, in our opinion, lead to problems, as log probability 0 implies probability 1. More natural choice might be to construct the language vectors using probability directly.

[2]Because of computational constraints, we cap the total number of lines per cluster to 20M. If the cluster corpus exceeds the total number of lines, we subsample the available data with $\alpha = 1.0$

[3]Here we slightly improve the methods of Chung and Liang. By following the original method as described above, the final size of the vocabulary will be lower than the target we set. This is because the cluster vocabularies V^{c_j} will contain overlapping tokens and merging will remove these duplicates. This negative effect becomes larger with the increasing number of clusters k. Therefore, on top of training the prescribed cluster vocabulary of size $|V^{c_j}|$, we also train slightly larger vocabularies $V_q^{c_j'}$ of size $|V_q^{c_j}| = q|V^{c_j}|$ for $q = 1.1, 1.2, 1.3$. We then select the minimum q that results in a vocabulary size of at least 120k tokens after merging the cluster vocabularies. After

After the tokenizers are trained, we merge the vocabularies using the method described in section 3.7.3.

The resulting clusters and the corresponding per-cluster vocabulary sizes are found in Tables 3.5, 3.6, 3.7, and 3.8.

By a quick inspection, we see that the clusters are often composed of languages that share a common script but are not necessarily related typologically. For example, we often see the Arabic and Urdu languages in the same cluster even though the languages belong to different language families.

Reproducing the VoCap method

From the high level, the VoCap method works by selecting the best vocabulary size for each language and then merging the monolingual vocabularies. The best vocabulary size is determined by maximizing the overall *Average Log Probability* metric defined in Equation 3.7.

To replicate the VoCap method, we first need to compute the ALP metric for each language l and each vocabulary size $V \in 1000, ..., 40\,000$. To that end, we train monolingual tokenizers using Sentencepiece with default settings for all 20 languages with vocabulary sizes from 1k to 40k.[4] We use 1M lines per language for training the monolingual tokenizers again, following Zheng et al. [11]. Note that for Chinese the tokenizer vocabulary size starts at 5k due to the large number of unique logograms in the language.

We then load a sample of CC100 data for each language (100k lines per language) and tokenize each monolingual corpus with the respective tokenizers of increasing vocabulary sizes. We are then able to compute the ALP(l, V) metric for each language l and each vocabulary size V.

Now we can proceed to greedily select the best vocabulary sizes. We start with selecting the lowest vocabulary size for each language (1k for all languages except Chinese where we start with 5k). We merge the selected vocabularies of the tokenizers as explained in section 3.7.3. Then in each iteration, we check which language would benefit the most from increasing the vocabulary size by 1000. Concretely, we check the increase in ALP for each language and increase the vocabulary size for that language by merging the bigger vocabulary with the total vocabulary. We repeat this process until the total vocabulary size reaches

that, we trim the vocabulary if needed. This improvement is done to ensure the final vocabulary size is exactly 120k tokens and the comparison between the methods is fair.

[4]The more effective way, not discussed by Zheng et al. [11], would be to modify the Sentencepiece unigram trainer code to produce a tokenizer after each prune iteration. That way we would get series of tokenizers with decreasing vocabulary size in one go, instead of running the trainer 40 times. As the computational cost of training the tokenizers is not high for our reduced set of 20 langugaes, we have not implemented this improvement.

Languages	Size
zh, ar, ru, bg	27425
el, ur, ta, te, th, he, ka	48841
en, es, tr, sw, vi, fr, de	43622
hi, mr	12108

(a) Chung et al. [10]

Languages	Size
el, zh, ar, ur, ta, te, th, he, ka	61285
en, es, tr, sw, vi, fr, de	43370
hi, mr	10370
ru, bg	16970

(b) Liang et al. [12]

Figure 3.5 Cluster assignments for 4 clusters

Languages	Size
el	7458
ru, bg	12975
ta, te	14318
en, es, tr, sw, th, ka, vi, fr, de	56509
ar, ur	13788
hi, mr	12030
zh	7458
he	7458

(a) Chung et al. [10]

Languages	Size
en, fr	12256
es, tr, sw, de	27342
ru, bg	16970
ar, ur	12256
vi	3770
hi, mr	10370
el, ta, te, th, he, ka	37713
zh	11313

(b) Liang et al. [12]

Figure 3.6 Cluster assignments for 8 clusters

Langs.	Size	Langs.	Size
ar, ur	14020	zh	7582
tr	7582	he	7582
en, fr	13549	ta	7582
hi, mr	12232	sw	7582
ru, bg	13194	ka	7582
vi	7582	th	7582
te	7582	es	7582
el	7582	de	7582

(a) Chung et al. [10]

Langs.	Size	Langs.	Size
de	8228	te	7200
ar	7200	vi	4113
en, fr	13370	el	10285
hi, mr	11313	ta	4113
ru, bg	18513	zh	12342
ka	8228	sw	6170
he	5142	es	7200
tr, th	14400	ur	6170

(b) Liang et al. [12]

Figure 3.7 Cluster assignments for 16 clusters

Languages	Size
ar	7200
bg	7200
de	7200
el	7200
en	7200
es	7200
fr	7200
he	7200
hi	7200
ka	7200
mr	7200
ru	7200
sw	7200
ta	7200
te	7200
th	7200
tr	7200
ur	7200
vi	7200
zh	7200

(a) Chung et al. [10]

Languages	Size
ar	7200
bg	8228
de	8228
el	10285
en	6170
es	7200
fr	7200
he	5142
hi	5142
ka	8228
mr	6170
ru	10285
sw	6170
ta	4113
te	7200
th	6170
tr	8228
ur	6170
vi	4113
zh	12342

(b) Liang et al. [12]

Languages	Size
ar	7000
bg	8000
de	8000
el	10000
en	6000
es	7000
fr	7000
he	5000
hi	5000
ka	8000
mr	6000
ru	10000
sw	6000
ta	4000
te	7000
th	6000
tr	8000
ur	6000
vi	4000
zh	12000

(c) Zheng et al. [11]

Figure 3.8 Allocated vocabulary sizes for 20 languages

120k tokens. Any tokens over the limit are removed from the vocabulary.

Contrary to Zheng et al. [11], we do not use the β rescaling factor to account for the pretraining corpus size (we describe the β parameter in related work subsection 2.4.2). By setting β to 0, we want to achieve the best ALP for each language regardless of its corpus size. This is because our goal is to balance the low-resource languages, not necessarily achieve the best performance on the downstream tasks.

The final vocabulary sizes for each language are found in Figure 3.8c

Merging the tokenizers

For all reproduced methods, the last step is to take several Unigram tokenizers and merge them into the final, multilingual tokenizer. Now we will describe how we merge tokenizers in our case. Unfortunately, the merging step is not described fully in any of the reproduced papers. In the case of the Unigram tokenizers, tokenizer τ consists of vocabulary (set of strings) $V_\tau \subset \Sigma^\star$ and the corresponding logits $L_\tau : \Sigma^\star \to \mathbb{R}$ so that $\sum_{t \in V_\tau} \exp(L_\tau(t)) = 1$. The logits are used for finding the most probable segmentation of an input sentence.

To create the merged vocabulary for input tokenizers $\tau_1, \ldots \tau_m$ we take the union over the vocabularies:

$$V_\tau := \bigcup_{i=1}^{m} V_{\tau_i} \tag{3.25}$$

We set the merged logits to the log of the average probability of the token in the input tokenizers:

$$\forall t \in V_\tau : L_\tau(t) := \log(\frac{1}{m} \sum_{i}^{m} \exp(L_{\tau_i}(t))) \tag{3.26}$$

If the token t is not present in some of the input tokenizers, we consider the probability of the token for that tokenizer to be zero.

In this way the sum of the probabilities of the tokens in the merged vocabulary is one and thus the merged tokenizer is a valid Unigram tokenizer. We can see this by the following derivation. We assume that the input tokenizers τ_i are valid Unigram tokenizers and thus the sum of the probabilities of the tokens in the input tokenizers is one:

$$\begin{aligned} \sum_{t \in V_\tau} \exp(L_\tau(t)) &= \sum_{t \in V_\tau} \exp(\log(\frac{1}{m} \sum_{i}^{m} \exp(L_{\tau_i}(t)))) = \\ = \sum_{t \in V_\tau} \frac{1}{m} \sum_{i}^{m} \exp(L_{\tau_i}(t)) &= \frac{1}{m} \sum_{i}^{m} \sum_{t \in V_\tau} \exp(L_{\tau_i}(t)) = \frac{1}{m} \sum_{i}^{m} 1 = 1 \end{aligned} \tag{3.27}$$

We argue that this is the most natural way to merge the tokenizers and so we assume this is probably the way the other authors did the merging. By observing the logits in the tokenizer released by Liang et al. [12][5], we see that the authors do merge the logits in some way but the sum of the probabilities in the final tokenizer is ≈ 4.55 (not counting the special tokens), which suggests some problems in the merging step. The tokenizer released by Zheng et al. [11] seems to be merged correctly[6].

[5]https://huggingface.co/facebook/xlm-v-base/blob/main/sentencepiece.bpe.model

[6]https://github.com/bozheng-hit/VoCapXLM/blob/main/VoCap_500k/sentencepiece.bpe.model

Chapter 4

Tokenizer properties affect the performance of language models

In this chapter, we propose and conduct an experiment to answer (**Q1**) how subword tokenizers differ in overlap and allocation of learned vocabularies. Moreover, we explore (**Q2**) which properties affect the language model representation quality. Our goal is to establish that the metrics we propose are useful for assessing the differences between tokenizers and that they are useful for comparing whether a given tokenizer is better than another.

To answer these questions, we train three tokenizers and assess how they differ in the metrics we propose. We then look at how are these differences manifested when we use these tokenizers to train otherwise identical multilingual models. We then assess whether the proposed metrics are good predictors of the model's performance.

The results presented in this section are selected from our paper Limisiewicz, Balhar, and Mareček [13]. We refer the reader to the paper for more in-detail analysis and additional experiments. Here we present experiments that are most relevant to the thesis goal.

4.1 Analysis of Tokenizer Properties

We train three distinct tokenizers — Huggingface Unigram, Huggingface BPE and TokMix tokenizer [13] described in the following subsection. We use the Huggingface implementation[1] of the training algorithms for the Unigram and BPE tokenizers. The training data for all tokenizers is the CC100 corpus sampled with the exponential smoothing factor $\alpha = 0.25$. After tokenizer training, we

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

perform an intrinsic evaluation of the tokenizers on each language separately and report the macro average of the metrics (For details see subsection 3.5.2).

After tokenizer evaluation, we use the tokenizers to train three masked language models. We use the same training data and the same model architecture for all models defined in subsection 3.7.1. The only difference is the tokenizer used to preprocess the data. We then evaluate the models using probing on three multilingual word-level tasks (part of speech tagging, dependency labeling and named entity recognition) and one sentence-level task (cross-lingual natural language inference). We evaluate the overall in-language performance and compare it to the tokenizer metrics. We then do a more fine-grained comparison by evaluating the model on each language separately and comparing the results to the tokenizer metrics also measured on each language separately. (For details see subsection 3.5.3)

4.1.1 TokMix tokenizer

To create our TokMix tokenizer, we run the Huggingface Unigram tokenizer training on each language corpus $C_l, l \in L$ separately. After this, we end up with 20 separate tokenizers with equal vocabulary sizes. We merge the vocabularies of all tokenizers into one large vocabulary $V = \bigcup_{l \in L} V_l$, average the token probabilities and trim the vocabulary to the desired size $|V| = 120\,000$.

4.2 Results

4.2.1 Intrinsic evaluation

In the Table 4.1 we see that the choice of the tokenization method largely influences the vocabulary allocation and overlap metrics. The Huggingface BPE tokenizer produces on average the longest tokens (high CPT), the most uniform allocation of tokens (high AR), and the least overlap between languages (high JSD). On the other hand, the Huggingface Unigram segments the text into shorter tokens and the average vocabulary overlap between all languages is much higher. The high overlap might be related to the low allocation as it is more likely that shorter tokens are shared between languages.

Interestingly, the TokMix tokenizer has a similar *vocabulary allocation* (CPT, AR) as the Huggingface BPE tokenizer. This is surprising as the TokMix tokenizer is based on the Huggingface Unigram tokenizer, which shows significantly lower scores.

Tokenizer	Alphabet	# UNKs	CPT	AR	JSD
Huggingface BPE, α=0.25	1000	14040.1	3.713	1253.7	0.783
TokMix, α=0.25	2497	1203.2	3.691	1163.4	0.773
Huggingface Unigram, α=0.25	12616	4.5	3.204	1010.5	0.745

Table 4.1 In the first batch of experiments, we compare the Huggingface Unigram, Huggingface BPE, and our TokMix method based on merging Huggingface Unigram tokenizers. Huggingface Unigram has significantly lower vocabulary allocation scores (-0.4 CPT and -153 AR) than BPE and TokMix. This means that Unigram uses shorter tokens and the capacity of the vocabulary is used less uniformly. Moreover, the vocabulary has more overlap between the languages for Unigram (JSD decrease by -0.03). The scores are macro averages over all languages, computed over a holdout portion of the CC100 corpus. We sample 10k lines from each language which we have empirically found to be enough to get representative results.

4.2.2 Extrinsic evaluation

Next, we use the tokenizers from Table 4.1 and pretrain three masked language models. We find that the choice of the tokenizer has a large impact on the model's in-language performance. The Table 4.2 shows that the Huggingface Unigram has a significantly lower performance than the other two tokenizers on the word-level tasks (NER, POS, UD). The overall performance on sentence-level NLI is similar across tokenizers.

We validate this observation by looking at the interaction between the tokenizer metrics and model performance on the level of individual evaluation languages Figure 4.1. In each scatterplot, each point represents a pair (τ, l) of a tokenizer τ from our three tested tokenizers and a language l from the set of languages available for the given task. The position of the point corresponds to the observed tokenizer metrics and task performance [2]. We see that the Huggingface Unigram tokenizer exhibits lower performance on the word-level tasks (NER, POS, UD) which corresponds to the lower vocabulary allocation metrics. We the most clear relationship between NER and CPT, UD and CPT, and POS and CPT. We quantify this relationship by computing Spearman's correlation coefficient between the tokenizer metrics and the model performance. The results are shown in Table 4.3. We see that the vocabulary allocation metrics are positively correlated with the model performance on the word-level tasks. The length of the tokens has a strong positive influence on POS, dependency labeling, and NER

[2] As explained in section 3.5.3, we center the tokenizer metrics and downstream task results by subtracting the mean for each language to account for the differences between languages we cannot control for. This way we see only the improvements and deteriorations of the tokenizers and models compared to the mean value for the given language.

results ($r > 0.65$), while it does not significantly affect NLI results. The correlation between the average rank and NER scores is weaker but still significant. Moreover, it is significantly correlated with XNLI accuracy with a medium coefficient $r = 0.56$. Our findings suggest that longer tokens and more tokens allocated for a given language in the vocabulary improve the in-language model performance on our tested tasks.

We conduct a similar analysis for the cross-lingual setting. We evaluate the probes on all languages except the one on which the probe was trained on. The results are shown in Figure 4.2 and a summary of the correlations in Table 4.4. Each point corresponds to a triplet $(\tau, l_{\text{src}}, l_{\text{tgt}})$ of a tokenizer τ, a language l_{src} the probe was trained on, and a language $l_{\text{tgt}} \neq l_{\text{src}}$ the probe was evaluated on. We see that the cross-lingual performance is positively correlated with the vocabulary overlap metric JSD in the case of the word-level tasks. This suggests that smaller vocabulary overlap between the languages improves the cross-lingual performance on the word-level tasks. On the other hand, we see that the correlation between the combined CPT metric and the cross-lingual performance is similarly strong and that there is also a strong correlation between JSD and combined CPT. In this case, the correlation between JSD and downstream performance could be caused by a confounding factor of better vocabulary allocation as measured by CPT.

Overall, we see that the differences between tokenizers are reflected in the representation quality. High CPT and AR metrics are correlated with better probe performance, especially on word-level tasks such as part of speech tagging, named entity recognition, and dependency labeling. Moreover, the cross-lingual performance is also correlated with the vocabulary allocation and overlap metrics.

Therefore, we see that our metrics are useful for assessing the differences between tokenizers. Moreover, the differences in tokenizers are reflected in the learned representations of the models, especially on word-level tasks.

Note that in this section we have included only a part of the experiments from our paper Limisiewicz, Balhar, and Mareček [13] published in ACL Findings 2023. Namely, we focus only on the experiments that are based on the same 20 languages as the rest of the thesis. For our purposes, we focus only on the more general conclusions. We refer the reader to the original paper for the full set of experiments, where we additionally discuss in more detail which tasks are more affected by which properties of the tokenizers.

Tokenizer	NER	POS	UD	XNLI
Huggingface BPE	66.3 $_{\pm 0.2}$	67.3 $_{\pm 0.4}$	54.5 $_{\pm 0.5}$	53.5 $_{\pm 0.3}$
TokMix	65.4 $_{\pm 0.3}$	66.5 $_{\pm 0.4}$	53.9 $_{\pm 0.5}$	52.3 $_{\pm 0.3}$
Huggingface Unigram	58.9 $_{\pm 0.2}$	54.0 $_{\pm 0.4}$	43.7 $_{\pm 0.4}$	53.2 $_{\pm 0.3}$

Table 4.2 Results of extrinsic evaluation for the Huggingface tokenizers. We observe significant changes for different tokenization methods. We report the F1 score for NER, POS and UD. XNLI is reported as accuracy. The scores are macro averages over all languages.

	V. Allocation	
	(AR)	(CPT)
CPT	**0.790**	-
NER	**0.394**	**0.657**
POS	0.320	**0.724**
Dep l.	0.266	**0.675**
NLI	**0.56**	0.388

Table 4.3 Spearman correlations between centered in-language task results and tokenizer measures. Statistically significant correlations ($p < 0.01$) are bolded.

	V. Overlap	**V. Allocation SRC**		**V. Allocation TGT**	
	(JSD)	(AR)	(CPT)	(AR)	(CPT)
NER	**0.553**	**0.172**	**0.412**	**0.409**	**0.568**
POS	**0.759**	**0.383**	**0.69**	**0.436**	**0.714**
Dep l.	**0.596**	**0.314**	**0.587**	**0.351**	**0.605**
NLI	-0.078	-0.039	-0.006	-0.083	-0.082
Retrieval	**0.156**	**0.214**	**0.139**	**0.214**	**0.144**

Table 4.4 Spearman correlations between cross-lingual transfer results and tokenization measures. vocabulary overlap is measured by JSD, we also measure the correlation with vocabulary allocation s of source and target language of the transfer directions. Statistically significant correlations ($p < 0.01$) are bolded.

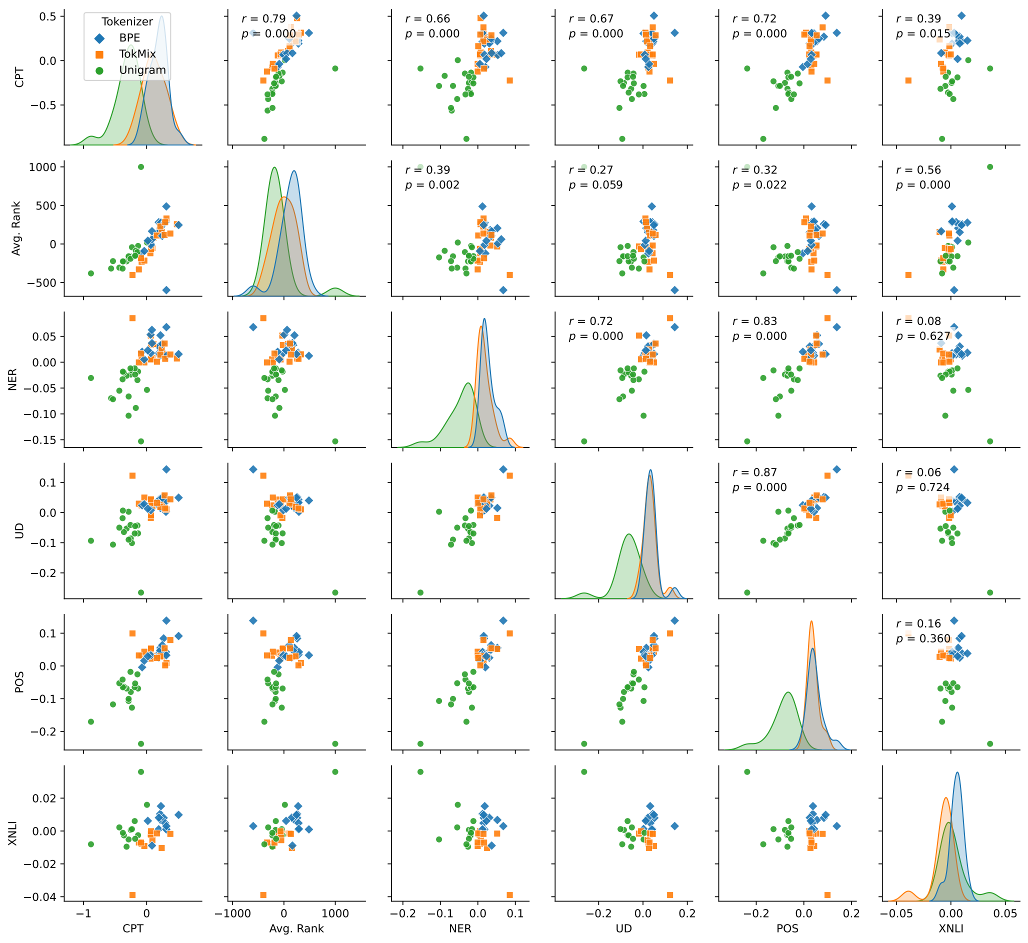


Figure 4.1 We compare the tokenizer metrics against the contextualized representation quality. For each tokenizer, we pretrain a masked language model, freeze it, and train a linear probe for each task and each of the available languages. We observe a high Spearman correlation between CPT and the word-level tasks (NER, POS, UD) and a high correlation between AR and the sentence-level task XNLI. This suggests that our vocabulary allocation metrics are good indicators of the tokenizer's quality and higher vocabulary allocation leads to better downstream performance. Each data point corresponds to an average result over three seeds of probe training and evaluating one of the languages. The results for each language are centered around the mean to account for the differences between languages as explained in section 3.5.3. The colors of points are assigned for specific tokenizers.

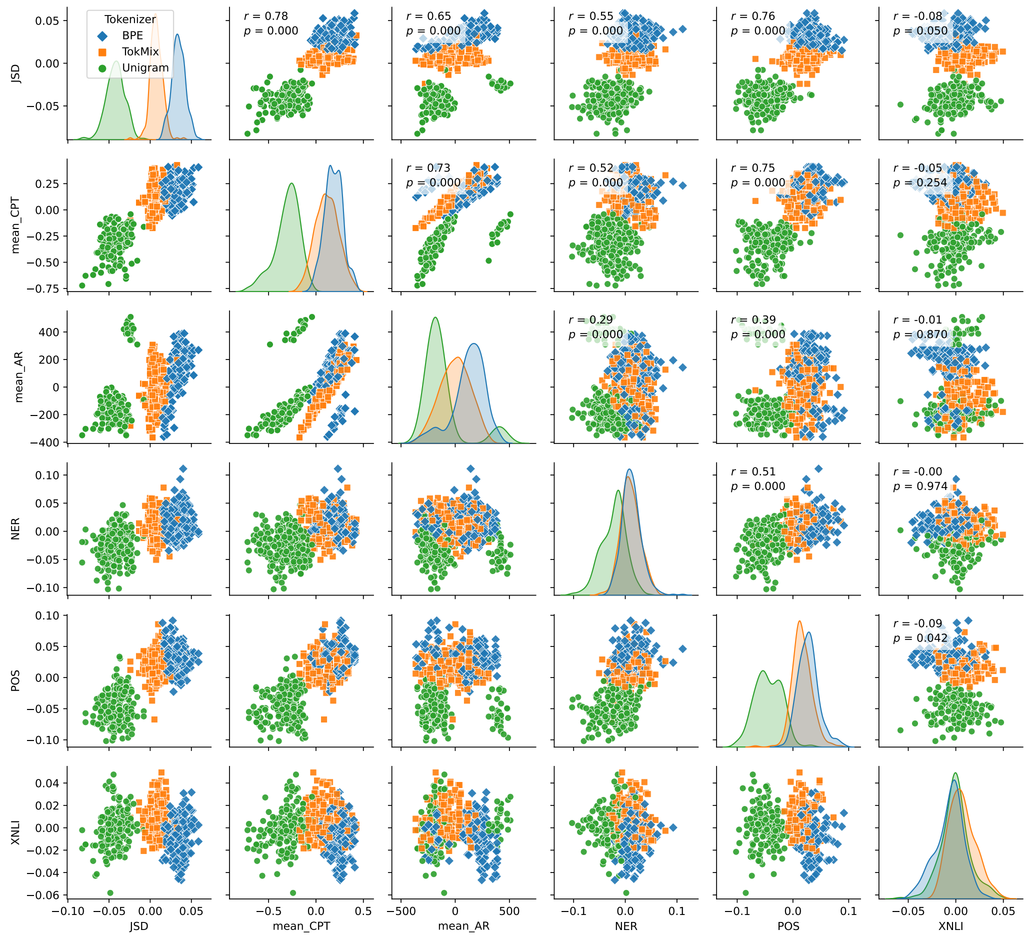


Figure 4.2 We compare the tokenizer metrics against the cross-lingual performance of the models. For each tokenizer, we pretrain a masked language model, freeze it, and train a linear probe on each of the available languages. Then we evaluate the models on all languages the probe has **not** been trained on, assessing the cross-lingual properties of the model. Here we observe a high correlation between JSD and the word-level tasks, especially the POS and UD. This suggests that less overlap (higher divergence) between the vocabularies of the languages leads to better cross-lingual performance for word-level tasks.

4.3 Findings

We find that the choice of the tokenization method largely influences the vocabulary allocation and overlap metrics (**Q1**). We see that Huggingface BPE better allocates the vocabulary overall and has a lower overlap between languages. On the other hand, the Huggingface Unigram tokenizer segments the text into shorter tokens and the average vocabulary overlap between all languages is much higher.

We find that the differences between tokenizers are reflected in the representation quality (**Q2**). High vocabulary allocation metrics are correlated with better probe performance, especially on the word-level tasks. Moreover, the cross-lingual performance is correlated with higher vocabulary allocation and lower overlap.

Our findings validate our proposed metrics as they are useful predictors of the model's performance.

Chapter 5

Design choices for better multilingual tokenizers

In this chapter, we propose and conduct a series of experiments to answer (**Q3**) what is the reason that the standard tokenizer training method does not work well in the multilingual setting. Our goal is to explore why was the Unigram algorithm found to be unsuitable for multilingual tokenization by our related work. Moreover, we investigate the reason behind the large difference in the quality of tokenizers in the previous Chapter 4.

To answer the question, we first investigate the differences in implementation between Huggingface Tokenizers[1] and the original Sentencepiece[2]. Next, we look at the influence of the training data size, the alphabet size, and finally the language imbalance in the training data. In the last experiment, we will examine closely the effect of the data imbalance on the tokenizer's quality per language.

5.1 Experiments

As shown in the previous Chapter 4, we observe that the Huggingface Unigram tokenizer leads to significantly worse metrics than the other tokenizers. We investigate this difference by turning to the original Sentencepiece implementation of the algorithm and running a comparable experiment. For comparison, we also train a comparable BPE tokenizer using the Sentencepiece library. We evaluate the tokenizers using our metrics.

Next, we train a series of Unigram tokenizers on different amounts of data and see how the data amount influences the tokenizers' quality. We sample $N = 1\,000, 10\,000, 100\,000, 1\,000\,000, 1\,500\,000, 2\,000\,000$ lines for each of the 20

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

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

languages, concatenate the samples and create a balanced corpora of different sizes. We then evaluate the tokenizers using our metrics.

We proceed with a similar analysis for the alphabet size. The alphabet of a tokenizer is the set of characters that are included in the vocabulary. We hypothesize, that too large alphabet size may influence the vocabulary allocation, we therefore use the character coverage parameter of the Sentencepiece library to control the alphabet size. The character coverage parameter determines, how many distinct Unicode characters are included in the vocabulary of the tokenizer. We train a series of Sentencepiece Unigram tokenizers with the character coverage parameter set to 98%, 99.5%, 99.95%, 99.995%, 99.9995%, and 100.0% on data sampled with $\alpha = 0.3$ from CC100.

Finally, for the data balance experiment, we train 5 tokenizers with an increasingly imbalanced corpus with $\alpha = 0.0, 0.3, 0.5, 0.7, 1.0$ sampled from the CC100 with 20 languages. [3] On one extreme we have the $\alpha = 1.0$, where all data available for each language is combined. On the other, we have $\alpha = 0.0$, where the data is sampled per line from each language with the same probability. We use the Sentencepiece Unigram tokenizer with the default settings. Specifically, we use the default character coverage of 99.95%. As usual, we evaluate the tokenizers on a balanced validation set sampled from a holdout portion of the CC100 corpus.

5.2 Results

5.2.1 Choice of implementation

The results are presented in Table 5.1. We see that the implementation has an effect on tokenization. We compare the Sentencepiece Unigram (α=0.25) tokenizer trained on the same data and with the same parameters (100% alphabet coverage) as the Huggingface Unigram (α=0.25). We see that there is a large difference between the two. The Huggingface Unigram underperforms all of our tokenizers. On the other hand, the Sentencepiece Unigram approaches the metrics of both BPE tokenizers. We further see that if we restrict the vocabulary size for the Sentencepiece Unigram ((α=0.3)), we close the gap between Unigram and BPE.

Interestingly, we observe that the Huggingface implementation of BPE seems to be better than the Sentencepiece implementation of BPE in our experimental setup, yielding higher vocabulary allocation metrics (CPT and AR).

[3] For $\alpha = 0.0, 0.3, 0.5, 0.7$ we make sure to sample at least 100k lines per language as we have found this to be important in subsection 5.2.2. We note that the data imbalance for $\alpha = 1.0$ is so large, that we needed to settle for 30k-70k training lines for the five least resourceful languages (ka, ur, te, mr, sw) because of memory constraints.

Tokenizer	Alphabet	# UNKs	CPT	AR	JSD
Huggingface BPE α=0.25	1000	14040.1	3.713	1253.7	0.783
Sentpiece. BPE α=0.25	1215	7235.6	3.666	1212.9	0.774
Sentpiece. Unigram α=0.3, 99.95% character coverage	2666	923.5	3.702	1190.7	0.768
Sentpiece. Unigram α=0.25, 100% character coverage	12577	4.5	3.629	1125.5	0.767
Huggingface Unigram α=0.25	12616	4.5	3.204	1010.5	0.745

Table 5.1 In this table, we compare the Huggingface and Sentencepiece implementations of the Unigram and BPE algorithms. The Huggingface Unigram tokenizer is a clear outlier in terms of all metrics. We can see that this is a problem in the implementation as the corresponding *Sentencepiece Unigram α=0.25, with 100% alphabet coverage* scores much higher on our metrics. Interestingly, we found that the BPE implementation (*Huggingface BPE α=0.25*) seems to be better in Huggingface than in Sentencepiece (*Sentpiece. BPE α=0.25*).

Because of the subpar implementation of the Unigram algorithm in the Huggingface library, we use the Sentencepiece implementation for the rest of the experiments.

5.2.2 Data size

We are interested in how much data is needed for the tokenizer training. We present the results in Table 5.2. We see that the metrics improve with the amount of data, but the improvement stops to be substantial after 100k-1M lines per language. We use these results as a rule of thumb for the rest of the experiments and where possible, we use at least 100k but preferably 1M lines per language.

5.2.3 Character coverage

We are also interested in the influence of the alphabet size on our metrics. As we have seen in Table 5.1, large alphabet size influences the vocabulary allocation metrics. We show the results in Table 5.3. We see that there is a direct relationship between the character coverage, alphabet size, and the number of unknown tokens in the validation set. We also see that our metrics are not largely affected by the alphabet size, as the alphabet accounts for at most 10% of the whole vocabulary size 120 000. The lowest vocabulary allocation metrics are on the extremes of the character coverage parameter, where the resulting alphabet size is either very small or very large. We assume that the small alphabet size leads to a

Lines per lang.	Alphabet	# UNKs	CPT	AR	JSD
1 000	3598	520.4	3.302	958.4	0.766
10 000	4725	117.8	3.598	1089.1	0.765
100 000	5041	65.5	3.696	1192.2	0.767
1 000 000	5079	62.6	3.702	1204.7	0.767
1 500 000	5176	55.9	3.705	1210.7	0.767
2 000 000	5180	56.4	3.705	1212.5	0.767

Table 5.2 We measure how much data is generally needed for the tokenizer training. We train handful of Sentencepiece Unigram tokenizers on different amounts of balanced multilingual data. We observe that after 100k-1M lines per language, the tokenizers converge to similar vocabulary allocation and overlap scores.

Coverage	Alphabet	# UNKs	CPT	AR	JSD
98.0%	539	17386.5	3.631	1115.3	0.749
99.5%	1136	7786.9	3.702	1173.1	0.765
99.95%	2678	910.6	3.705	1196.7	0.768
99.995%	4813	83.0	3.695	1188.7	0.769
99.9995%	8226	10.2	3.678	1164.2	0.769
100.0%	13658	1.9	3.650	1124.1	0.768

Table 5.3 We check the tradeoff of including a large alphabet size. We train Sentencepiece Unigram tokenizers with a different target character coverage and observe the resulting alphabet size, number of UNKs and tokenizer metrics. We observe that the alphabet size grows with the coverage and the number of UNKs decreases, as expected. We observe that at both extremes of the character coverage parameter, the vocabulary allocation decreases. The results indicate that the alphabet size between 1000 and 5000 provides a good tradeoff between the number of UNKs and the allocation metrics, while including all characters in the alphabet does not come with a significant decrease in the allocation metrics (-0.05 CPT).

Tokenizer	Alphabet	# UNKs	CPT	AR	JSD
Unigram α=0.0	2975	617.1	3.712	1212.9	0.767
Unigram α=0.3	2666	923.5	3.702	1190.7	0.768
Unigram α=0.5	2859	729.0	3.618	1143.8	0.769
Unigram α=0.7	2733	883.2	3.556	1107.1	0.770
Unigram α=1.0	2476	1286.3	3.442	1041.8	0.772

Table 5.4 We train five Sentencepiece Unigram tokenizers on increasingly imbalanced multilingual dataset. We see that the macro averaged metrics decrease with the increasing imbalance, suggesting that on average, the tokenizer represents the languages less well.

large number of unknown tokens and the tokenizer is forced to segment words containing characters outside of the alphabet, as these unknown tokens might even be characters with diacritics. In the range of 1000-5000 alphabet size, we see that the metrics are not largely affected by the alphabet size. On the other extreme, where alphabet size is large, we suspect that the alphabet starts to take up a larger portion of the vocabulary and the tokenizer has less capacity for longer tokens which we observe as a lower overall CPT and AR. We note that the observed differences are small and including all characters in the training set does not come with a large decrease in our tokenizer metrics (-0.05 CPT and -70 AR compared to 99.95% coverage). For later experiments, we use the Sentencepiece default character coverage of 99.95%. When comparing tokenizers with different alphabet sizes, we are aware of the fact that the metrics might be affected by the alphabet size and we take this into account when interpreting the results.

5.3 Data imbalance

Finally, we investigate, how the training data imbalance between high-resource and low-resource languages affects the tokenizer performance.

The results in Table 5.4 demonstrate a clear disparity in the quality of tokenization depending on the data balance. Training on balanced data leads to higher overall metrics than on unbalanced data [4]. The imbalance also affects the alphabet size as it is possible to cover 99.95% of the characters in the training data with a smaller alphabet because of the overrepresentation of a few high-resource languages.

[4]This is naturally affected by the evaluation scheme where we weight each language equally. The equal weighting is in line with our goal of improving the tokenization for all represented languages as explained in subsection 3.5.2

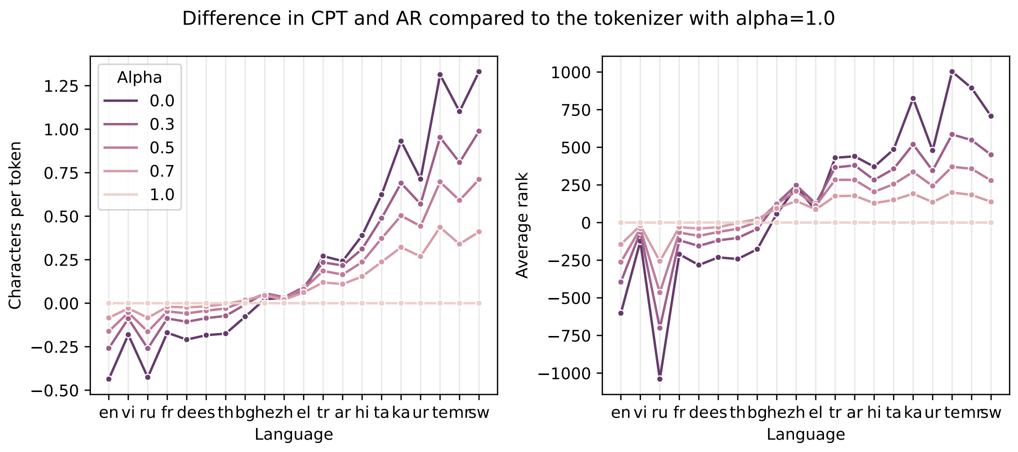


Figure 5.1 We examine the impact of the language imbalance on the Sentencepiece Unigram tokenizer training. We train five tokenizers with an increasing language imbalance controlled by the α parameter. Then we look at the effect on the vocabulary allocation metrics per language. We subtract the results using the most unbalanced tokenizer with $\alpha = 1.0$. As expected, the more balanced tokenizers have higher vocabulary allocation scores for low-resource languages and lower scores for high-resource languages. Interestingly, the effect varies across languages. For example, the vocabulary allocation of high-resource Vietnamese or French is not as affected by the decrease in training data as English or Russian.

We explore the reason for the decreasing performance of the tokenizers trained on imbalanced data in Figure 5.1. We plot the differences in vocabulary allocation metrics between the most unbalanced tokenizer $\alpha = 1.0$ and the rest of the tokenizers. We sort the languages by the data size available starting with the highest-resource languages and ending with the low-resource. We see that the vocabulary allocation metrics for the high-resource languages (en, vi, ru, fr, es, th) are decreasing with the increasing data balance between the languages, compared with the unbalanced baseline. On the other hand, the low-resource languages (ka, ur, te, mr, sw) are disproportionally more improved by the increasing balance and their vocabulary allocation metrics are increasing significantly. This suggests that the marginal benefit of adding more data to the high-resource languages is lower than the incurred cost on the quality of tokenization for the low-resource languages. We see the result of this tradeoff in the Table 5.4 as an overall decrease in the average CPT and AR.

We hypothesize, that a possible reason that the standard method of training a tokenizer on a joint corpus was reported to not work by the existing works [7, 10, 11, 12] might be the data imbalance in the training data. Rust et al. [7] finds the mBERT tokenizer trained with $\alpha = 0.7$ inadequate to represent low-resource languages. Chung et al. [10] and Zheng et al. [11] use the Unigram baseline with $\alpha = 0.7$. While Liang et al. [12] uses a baseline with $\alpha = 0.5$. All balancing methods report the most substantial improvements on the low-resource languages. In the next chapter, we will therefore compare the balancing methods with the Sentencepiece Unigram trained on balanced and unbalanced data.

5.4 Findings

In this chapter, we have investigated **Q3:** What is the reason that the standard tokenizer training method does not work well in the multilingual setting? To this end, we explore how different design choices affect the quality of the tokenizers.

We find that the implementation of the Unigram algorithm in the Huggingface library is subpar and that the Sentencepiece implementation yields better results.

We observe that we need around 100k-1M lines per language to train a good, multilingual tokenizer.

The alphabet size affects the number of UNK tokens but does not have a significant influence on the rest of the metrics if we stay in the range of 1000-5000 alphabet size.

Most importantly, we observe that tokenizer training data imbalance influences the per-language metrics heavily and that it lowers the tokenization quality for the low-resource languages more than it improves it for the high-resource.

Chapter 6

Vocabulary balancing methods

Finally, we replicate the works of Chung et al. [10], Zheng et al. [11], and Liang et al. [12] (we colectivelly address these works as the "vocabulary balancing methods") and create several variations of their tokenizers that aim to improve the text segmentations, especially for the low-resource languages. We carefully compare these replicated tokenizers with the traditional method of training Sentencepiece Unigram tokenizers on a joint, multilingual corpus.

By comparing the balancing tokenizers, we aim to answer (**Q4**) what is the effect of using the balancing methods on the representation of low-resource languages? And (**Q5**) how do the balancing methods compare to the standard method of training the tokenizer on balanced and unbalanced joint corpus?

Our method is to replicate and train all examined tokenizers. Then we perform the intrinsic and extrinsic evaluation of the tokenizers. We examine closely the overall perfomance, as well as the changes for the individual languages.

6.1 Experiments

Our experimental setup consists of comparing the vocabulary balancing tokenizers by Chung et al. [10], Zheng et al. [11], and Liang et al. [12] with the standard Sentencepiece Unigram. We train the Unigram tokenizer on imbalanced and balanced data as we have found this to be an important parameter that influences the representation of low- and high- resource languages(section 5.3). For additional context, we also add the tokenizers from Chapter 4 for assessing the overall tokenizer metrics.

To answer **Q4** and **Q5**, we assess the tokenizers using our proposed *vocabulary allocation* and *vocabulary overlap* metrics. We look at the overall statistics and also more closely on the metrics computed per language. By assessing the metrics per language, we examine what is the effect of the balancing methods on the quality of representation of low-resource languages. Moreover, by comparing the balancing methods with the standard Sentencepiece Unigram, we assess how are the approaches related and how they differ.

After the intrinsic analysis, we select a representative subset of the reproduced tokenizers and compare them extrinsically by training masked language models. We evaluate the models in the in-language and cross-language setting on a selection of downstream tasks and assess the overall differences between tokenizers. We further assess the (**Q4**) and investigate what is the effect of the balancing methods on the performance of the models across languages. We also further assess the (**Q5**) and compare the balancing methods with the standard Sentencepiece Unigram in the extrinsic evaluation setting.

In this chapter, we compare the following tokenizers:

- **Sentencepiece Unigram** with $\alpha = 0.0, 0.3, 0.5, 0.7, 1.0$ (see section 5.3)
- **Chung et al. [10]** with 4, 8, 16, 20 clusters
- **Zheng et al. [11]** with the maximized ALP metric
- **Liang et al. [12]** with 4, 8, 16, 20 clusters

For additional context, we also include the following tokenizers from Chapter 4 in the overall intrinsic comparison of the tokenizers:

- **Huggingface BPE** with $\alpha = 0.25$
- **Huggingface Unigram** with $\alpha = 0.25$
- **TokMix** with $\alpha = 0.25$
- **Sentencepiece BPE** with $\alpha = 0.25$

We note that for Sentencepiece Unigram with $\alpha = 0.0$ and 0.3 we retrain the tokenizers on 20M and 10M lines of data respectively (compared to 2M and 5M from Table 5.4) to match the data provided to the replicated, balancing methods. This does not increase the metrics by a lot as observed in subsection 5.2.2. Nevertheless, we decided to match the data sizes between the balancing methods and the Unigram tokenizers.

For the extrinsic evaluation, we select 6 tokenizers. Namely, we select the balancing methods by Chung et al. [10] and Zheng et al. [11]. For the Chung clustering method, we select a low- and high- number of clusters $k = 4$ and 16. We compare the selected balancing tokenization methods to standard Unigram tokenizers trained on differently balanced datasets with $\alpha = 1.0, 0.3$ and 0.0.

6.2 Results

6.2.1 Comparison of balancing methods

In Table 6.1, we compare overall metrics for all tokenizer experiments. We sort the table by the CPT metric. At the extremes, we see that the unbalanced Unigram tokenizers ($\alpha = 1.0, 0.7$) are placed at the bottom of the results along with the underperforming Huggingface Unigram implementation. On the other hand, we see that generally, the standard tokenizers trained on a more balanced dataset ($\alpha = 0.3, 0.0$) provide good results on the CPT metric.

Next, we see that the Zheng method and clustering methods of Chung and Liang with 20 and 16 clusters are close to the best tokenizers in terms of CPT. On the other hand, clustering methods with a lower number of clusters are closer to the unbalanced Unigram tokenizers ($\alpha = 1.0, 0.7$).

We visualize our vocabulary allocation metrics on a scatterplot in Figure 6.1 to better observe the differences between the tokenizers and explore the relationship between CPT and AR. Each point on the scatterplot is one tokenizer and its position is determined by the overall CPT and AR metrics. We connect related experiments with a line and color-code them. Here we can see that the tokenizers with high CPT often have high AR. Nevertheless, the balancing methods seem to have generally lower AR while having a comparable CPT to the other methods[1]. With the context of different tokenization methods, we can see the degree of Huggingface Unigram's underperformance.

Crucially, we see that the reproduced methods of Chung et al. [10], Zheng et al. [11], and Liang et al. [12] do improve over the unbalanced baselines $\alpha = 1.0, 0.7, 0.5$ on the CPT metric, especially with a higher number of clusters. On the other hand, they do not outperform the simple case of training the Sentencepiece Unigram on a balanced dataset $\alpha = 0.0$. We also observe that the clustering methods with a higher number of clusters along with Zheng are close to each other on the CPT-AR plot. We assume this is because, with higher k, the clustering methods reduce to the Zheng method (training separate tokenizers for each language). Similarly, with a lower number of clusters, the methods are much closer to the vanilla Sentencepiece Unigram $\alpha = 1.0$ trained on the unbalanced dataset. We hypothesize that with a low amount of clusters, the original data imbalance starts to degrade performance for low-resource languages inside the clusters.

We also explore the relationship between vocabulary allocation and vocabulary overlap on the Figure 6.2. We see that the differences between all methods based on Sentencepiece are small compared to the differences between Hugging-

[1]We also observe that there are no tokenizers with low CPT but high AR Our intuition is that it is not possible to construct a tokenizer with a high number of useful tokens which are all very short.

Tokenizer	Alphabet	# UNKs	CPT	AR	JSD
Hugg. BPE, α=0.25	1000	14040.1	3.713	1253.7	0.783
Unigram, α=0.0	2975	617.1	3.712	1212.9	0.767
Chung 20 clusters	4123	270.3	3.702	1098.7	0.766
Unigram, α=0.3	2666	923.5	3.702	1190.7	0.768
TokMix, α=0.25	2497	1203.2	3.691	1163.4	0.773
Chung 16 clusters	3933	387.1	3.677	1102.2	0.767
Liang 20 clusters	3709	341.4	3.676	1103.2	0.765
Zheng 20langs	4854	245.7	3.673	1094.5	0.765
Liang 16 clusters	3655	416.8	3.669	1106.2	0.767
Sentpiece. BPE, α=0.25	1215	7235.6	3.666	1212.9	0.774
Unigram, α=0.5	2859	729.0	3.618	1143.8	0.769
Chung 8 clusters	4870	684.4	3.575	1061.1	0.770
Unigram, α=0.7	2733	883.2	3.556	1107.1	0.770
Chung 4 clusters	3253	648.6	3.546	1071.9	0.768
Liang 8 clusters	4283	568.2	3.544	1081.6	0.767
Liang 4 clusters	3698	419.2	3.512	1082.5	0.769
Unigram, α=1.0	2476	1286.3	3.442	1041.8	0.772
Hugg. unigram, α=0.25	12616	4.5	3.204	1010.5	0.745

Table 6.1 In this summary table, we present all tokenizers used in this chapter. In the table, we include the tokenizers obtained by replicating the papers Chung et al. [10], Zheng et al. [11], and Liang et al. [12] in our setting. We also include the Huggingface tokenizers from Table 4.1 and Sentencepiece Unigram tokenizers from Table 5.4. As we can see, the Huggingface Unigram tokenizer is a clear outlier in terms of all metrics even after taking in account the higher alphabet size as explored in Table 5.3. Further, we can see that the clustering methods with a higher number of clusters are improving over the baselines the authors used (*Unigram, α=0.5* and *Unigram, α=0.7*). On the other hand, we see that using more balanced data for training the Sentencepiece Unigram (*Unigram, α=0.0*) leads to better overall performance compared to the replicated methods. We note that the alphabet sizes for all relevant tokenizers stay in the stable range of 1000-5000 so we do not expect this variable to influence the tokenizer metrics. The rows are sorted by the CPT score.

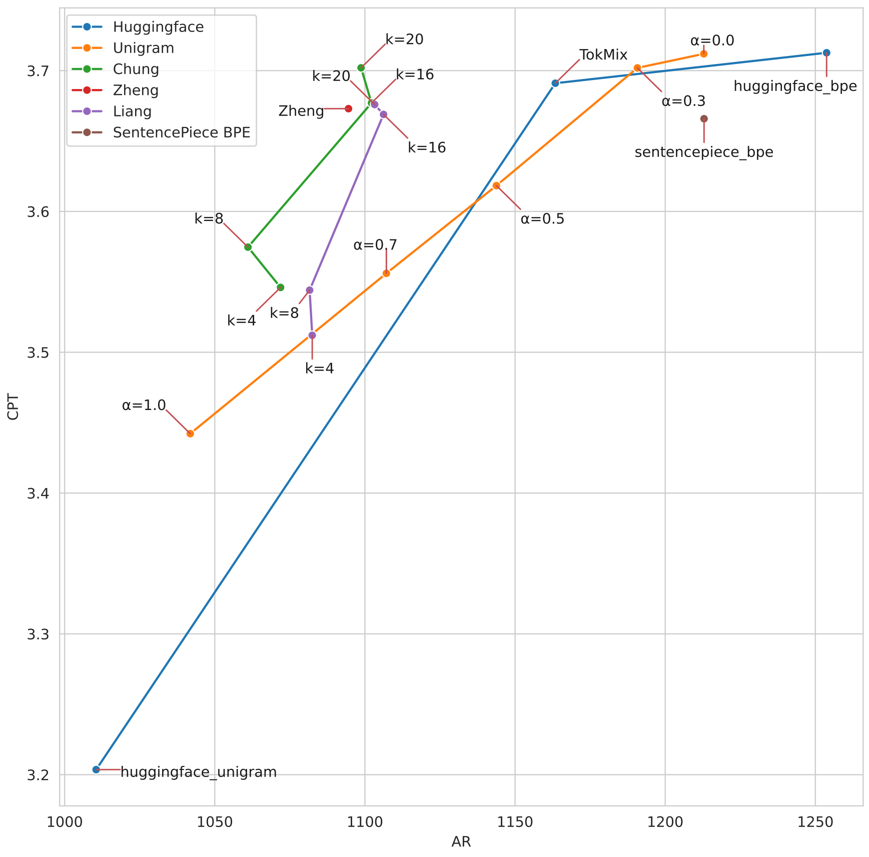


Figure 6.1 We visualize the overall vocabulary allocation metrics for all tokenizers from Table 6.1. We observe that the vocabulary allocation scores are related — higher AR usually means higher CPT. We also observe that Huggingface Unigram is a clear outlier, although a combination of separate, monolingual Huggingface Unigrams (TokMix) approaches the performance of the Sentencepiece Unigram with the corresponding data imbalance ($\alpha = 0.3$). We see that the balancing methods overperform the unbalanced Unigrams ($\alpha = 1.0$, $\alpha = 0.7$) in terms of CPT but perform similarly or worse to the simple case of running the Sentencepiece Unigram trainer on a balanced set $\alpha = 0.0$.

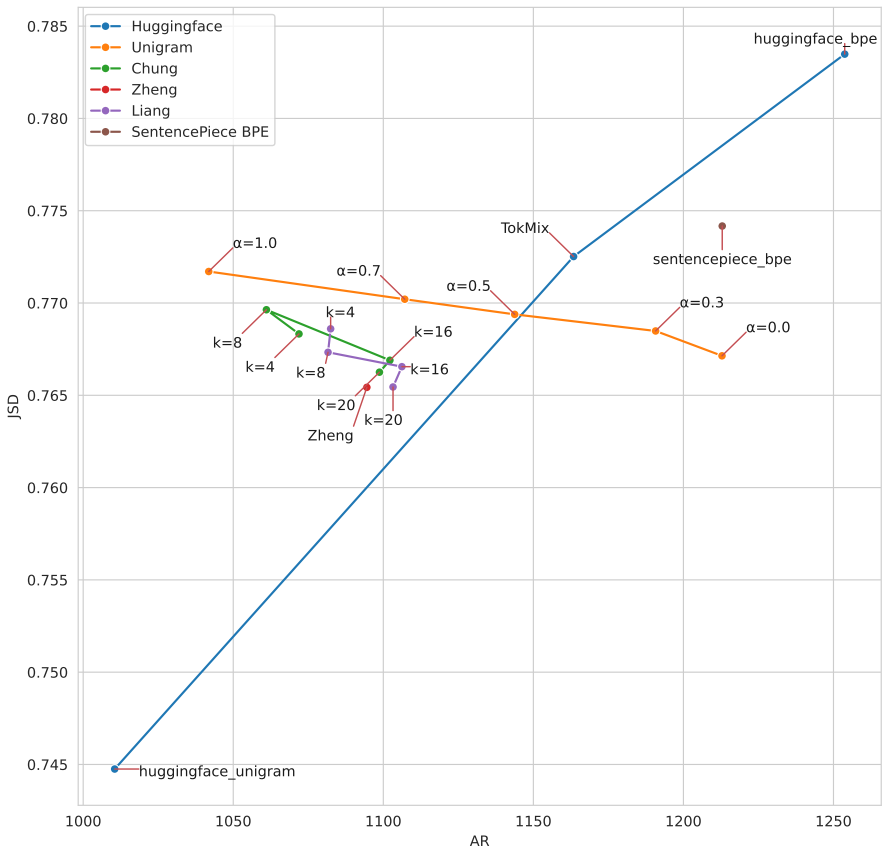


Figure 6.2 We visualize the tokenizers from Table 6.1 in terms of Average Rank and Jensen-Shannon Divergence. Here we can see that all methods based on Sentencepiece result in similar overlap independent of the allocation. This is interesting because the replicated balancing methods (Chung, Zheng, Liang) work by splitting the data and training separate tokenizers. Nevertheless, after merging the separate subtokenizers they all seem to end up with similar vocabulary overlaps. The highest vocabulary isolation is surprisingly achieved by the Huggingface BPE tokenizer, which is contrary to the hypothesis stated by Chung et al. [10] and Zheng et al. [11] that the tokenizers trained on the concatenation of all data tend to select subwords shared across all languages.

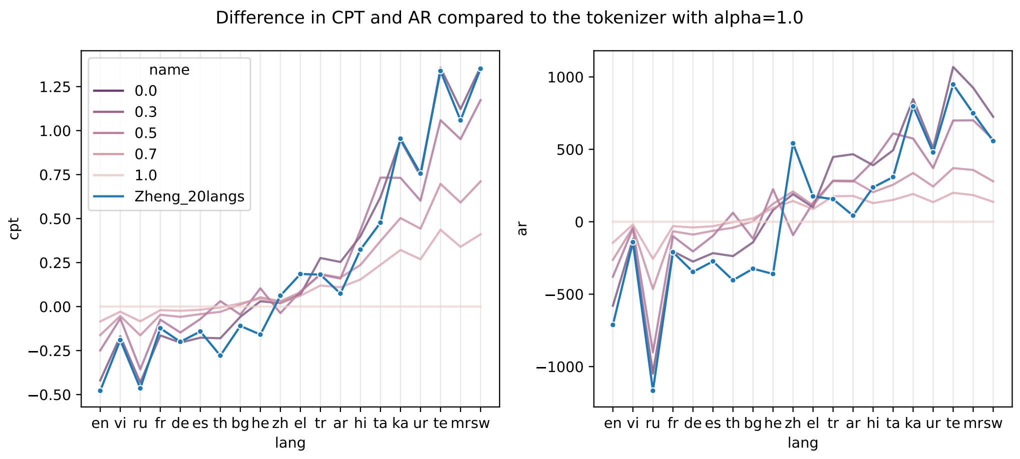


Figure 6.3 We zoom into the results of the Zheng method and compare the vocabulary allocation across the individual languages represented by this tokenizer against the backdrop of the vanilla Unigram tokenizers trained with different data imbalances from 5.1. We observe a striking similarity between the vocabulary allocation of the Zheng tokenizer and the Unigram tokenizer with $\alpha = 0.0$, especially in terms of characters per token. This comes as a large surprise because the Zheng method works by training a separate tokenizer for each language and then merging them. Despite the different methods of obtaining the vocabulary, the resulting tokenizers are very similar across the languages.

face tokenizers. This is surprising because the assumption shared by all authors of the balancing methods is that by training separate tokenizers, we achieve lower overlap between unrelated languages. Nevertheless, we see that the overall overlap is largely similar and more influenced by the choice of implementation.

If we ignore the Huggingface outliers and interpret only the differences in the overlap between the balancing methods, we still see a counterintuitive trend where the Zheng method and a higher number of clusters have a *larger overlap in vocabulary between languages* (lower JSD) than the standard tokenizers and clustering methods with lower number of clusters.

6.2.2 Comparison of balancing methods per language

We investigate the differences between the balanced Unigram and the replicated methods in more detail by examining the CPT and AR metrics computed **per language**. This way we can better assess (**Q4**) how the methods improve the low-resource languages. We also more closely compare (**Q5**) the balancing methods with the Unigram tokenizers trained on different data imbalances. We do this by plotting only the differences per language between the methods and the Unigram

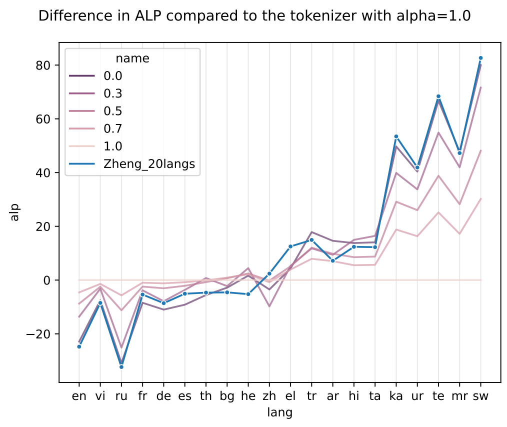


Figure 6.4 Intrigued by the similarity between the Zheng tokenizer and the Unigram tokenizer with $\alpha = 0.0$ from Figure 6.3 we also look at the ALP metric which is used for the selection of vocabulary sizes in the Zheng method. Here we see that the greedy optimization of ALP across languages indeed results in a similar vocabulary allocation as the Unigram tokenizer with $\alpha = 0.0$.

tokenizer with the highest imbalance $\alpha = 1.0$ similar to what we did in Figure 5.1.

We start by comparing the Zheng method with the increasingly imbalanced Unigram tokenizers in Figure 6.3. We plot the increase or decrease in vocabulary allocation metrics for each language sorted by the data available. Remarkably, we see that the Zheng method is strikingly similar in terms of CPT and AR per language to the Unigram tokenizer trained on the balanced set $\alpha = 0.0$. The similarity seems to be higher in the CPT metric although the AR metric is also similar especially for the highest and lowest resource languages. We find this observation quite surprising because of the distinctness of the Zheng method — it trains a separate tokenizer for each language and then merges the vocabularies together. Nevertheless, the resulting tokenizer is very similar to the Unigram tokenizer trained on the joint, balanced set.

Intrigued by the similarity between the Zheng tokenizer and the Unigram tokenizer with $\alpha = 0.0$, we also look at the differences in ALP metric which is used for the selection of vocabulary sizes in the Zheng method. In Figure 6.4 we see that the greedy optimization of ALP across languages indeed results in a similar vocabulary allocation as the Unigram tokenizer with $\alpha = 0.0$.

Next, we inspect the Chung method and compare it in detail to our Unigram tokenizers in Figure 6.5. For comparison, we select a run with a low number of clusters (k=4) and a high number of clusters (k=16). We see that the different numbers of clusters yield different results. In the case of a higher number of clusters, we see that the tokenizer exhibits a similar trend in CPT and AR across the languages as the balanced Unigram tokenizer with $\alpha = 0.0$ albeit with some

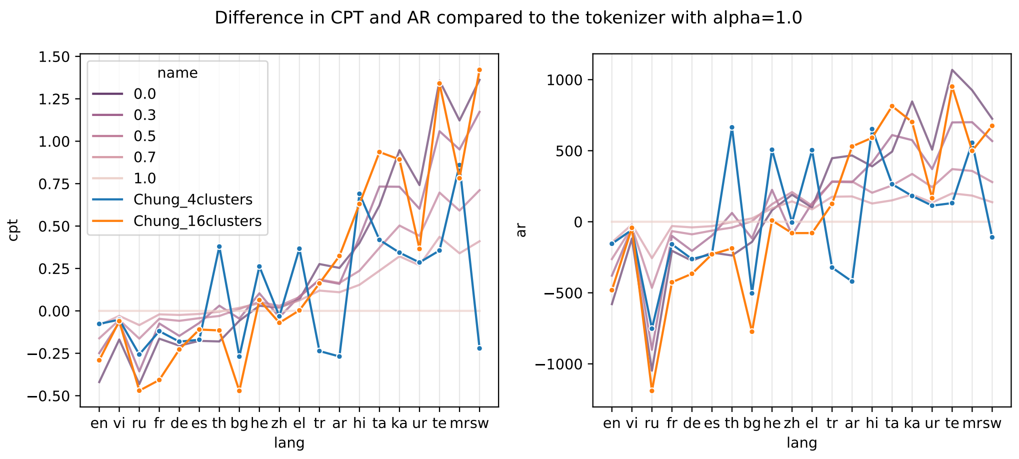


Figure 6.5 Here we inspect the language-level vocabulary allocation of the Chung method. We see that the Chung method with higher number of clusters (k=16) resembles the Unigram tokenizer with $\alpha = 0.0$ with large drops in allocation for Bulgarian (bg), Urdu (ur), Marathi(mr) and to a lesser degree French (fr). The lower number of clusters (k=4) differs more from the Unigram tokenizers. We see large increases in allocation for Thai (th), Hebrew (he), Greek (el), and Hindi (hi) and large decreases for Bulgarian (bg), Turkish (tr), Arabic (ar), and Swahili (sw).

deviations. In the case of a lower number of clusters, the metrics per language seem to be more distinct compared to our Unigram tokenizers.

We look at the CPT per language for k=16 more closely and identify the languages where the Chung tokenizer differs significantly from the Unigram tokenizer with $\alpha = 0.0$. We see that the CPT drops significantly for Bulgarian (bg), Urdu (ur), Marathi (mr), and to some degree French (fr). On the other hand, we see smaller improvements for English (en), Vietnamese (vi), Spanish (es), Thai (th), Hindi (hi), and Tamil (ta). We compare this to the cluster assignments in Figure 3.7a. Revealingly, we observe that all languages with the large drop in CPT have been assigned to a cluster with another, higher-resource language. Bulgarian is assigned with Russian (8th largest corpus versus 3rd largest corpus), Urdu with Arabic (17th vs. 13th), Marathi with Hindu (18th vs. 14th) and French with English (4th vs. 1st). This seems to be in line with our hypothesis that inside clusters, the data imbalance may hurt the representation of low-resource languages.

We continue with a similar analysis for the 4 clusters. We see that the CPT for Bulgarian (bg), Turkish (tr), Arabic (ar), and Swahili (sw) is lower than any of our Unigram tokenizers. On the other hand, we see significant improvements for Thai (th), Hebrew (he), Greek (el), and Hindi (hi) over our Unigram tokenizers. Additionally, Marathi (mr) achieves the highest CPT increase over the unbalanced

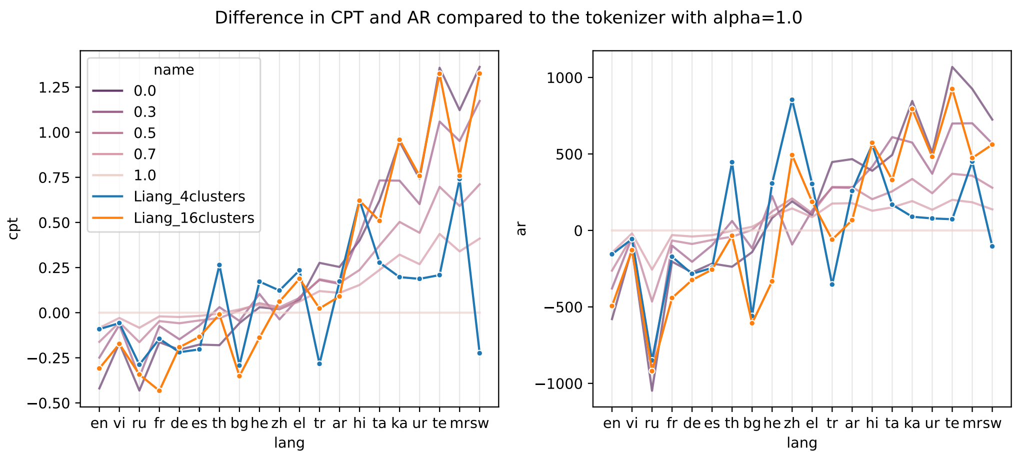


Figure 6.6 We inspect the language-level vocabulary allocation of the Liang method. We see similarities to the Chung method in Figure 6.5. The main differences seem to be the improved Chinese and Arabic for 4 clusters and worse Hebrew and better Urdu for 16 clusters. Overall the results are similar.

Unigram baseline, although the increase stays in the range of the $\alpha = 0.5$ Unigram tokenizer. We again look at the cluster assignments in Figure 3.5a. We observe that Bulgarian and Arabic are assigned to a cluster with higher-resource Russian (3rd) and Chinese, which could explain the decrease in CPT for the two. Similarly, Swahili and Turkish which use the Latin script are assigned to a cluster with higher resource English (largest corpus) and Vietnamese (2nd largest). On the other hand, we see that Thai, Hebrew, and Greek are assigned to a cluster with lower-resource languages — Tamil (ta), Georgian (ka), Urdu (ur), and Telugu (te). As we have observed, Thai, Hebrew, and Greek benefit from this assignment while the lower-resource languages seem to have a lower CPT than the $\alpha = 0.7$ Unigram and therefore approach the unbalanced baseline.

We look at the Liang et al. [12] replication results in Figure 6.6. We see that despite a slightly different clustering method and per-cluster vocabulary size selection, the Liang method exhibits similar patterns we observed in the Chung method.

Overall, we infer that the Chung and Liang methods are sensitive to cluster assignments. Because the training data are merged per cluster, if a low-resource language gets assigned to a cluster with a high-resource language, the language imbalance acts in favor of the high-resource language. As we know from our experiments in section 5.3 presented previously, the benefit of adding more data to the high-resource language is lower than the cost that incurs on the low-resource language. We believe this is the cause of the lower overall CPT and AR for the clustering methods.

6.2.3 Comparison of balancing methods on downstream tasks

We validate our observations from the previous subsection 6.2.2 by evaluating the tokenizers extrinsically. We again address (**Q4**) and (**Q5**) and investigate the differences between the balancing methods and the Sentencepiece Unigram tokenizers. In this section, however, we focus on the actual influence of the tokenizers on the multilingual language models and assess the differences between tokenizers by the performance on downstream tasks.

We select the balancing methods by Chung et al. [10] and Zheng et al. [11]. For the clustering method, we select a low- and high- number of clusters $k = 4$ and 16. We compare the selected balancing tokenization methods to the standard Unigram tokenizers trained on differently balanced datasets with $\alpha = 1.0, 0.3$ and 0.0

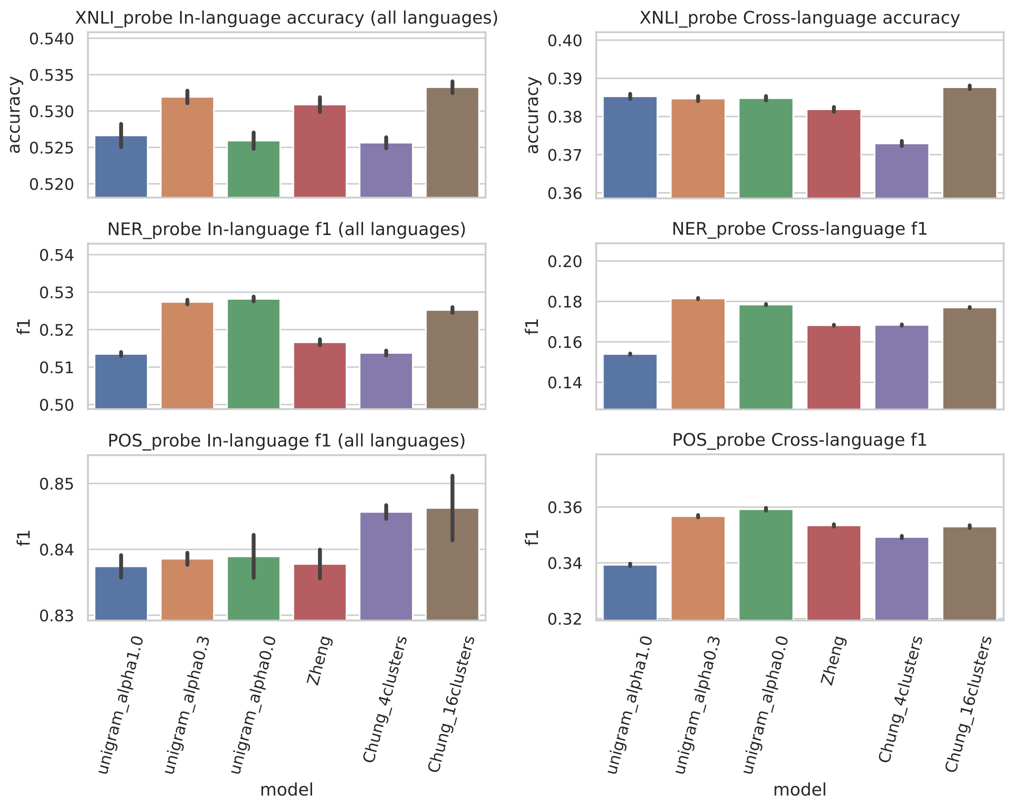
Overall in-language and cross-language results

In Figure 6.7, we report the overall in-language and cross-language results for the models. We observe the clearest regularity in cross-language performance for word-level tasks (NER and POS), where all balancing methods and Unigram $\alpha = 0.0, 0.3$ improve over the unbalanced $\alpha = 1.0$ model. Next, we see higher POS in-language scores for the Chung methods and higher NER in-language scores for the balanced unigrams ($\alpha = 0.0$ and 0.3). For in-language NLI, we do not see any systematic effect — the differences between the models are small (< 1 percentage point) and the outlier behavior of $\alpha = 0.3$ compared to $\alpha = 1.0$ and 0.0 suggests that the variance is caused by the finetuning rather than any tokenizer effect.

In-language and cross-language results examined by language

We inspect the results more closely on the language level. We again compare the performance of the more balanced models to the unbalanced model by plotting the difference in accuracy and F1 score for the tasks by language. The languages are again sorted in descending order based on the amount of available data.

In Figure 6.8, we see the in-language results laid out by language. We see a large effect of the probe training language on the NER F1 score and to some degree an effect on the POS F1. We do not see a systematic effect on NLI. For NER, we see the effect of the tokenizer language balance reminiscent of the results in Figure 6.3 and Figure 6.5. For high-resource languages, we observe a decrease in performance across the board. This is counterbalanced by a larger increase in performance for the low-resource languages. This effect seems to be largest for the $\alpha = 0.0$ Unigram tokenizer, Zheng method and Chung with 16 clusters. For $\alpha = 0.3$ Unigram and Chung with 4 clusters, we see a similar effect but with a



(a) In-language results **(b)** Cross-language results

Figure 6.7 We select the replicated methods by Chung et al. [10] and Zheng et al. [11] and compare them with the vanilla Unigram tokenizers. For comparison, we choose the unbalanced Unigram tokenizer with $\alpha = 1.0$ and then two stronger baselines with $\alpha = 0.0$ and $\alpha = 0.3$ trained on more balanced data. We then pretrain masked language models that differ only in the tokenizer they use and assess the performance of these models on the downstream tasks using probing. We test two settings — in-language performance, where the model is trained on each of the available languages and then evaluated on the same language, and cross-language performance, where the model is also trained on each language but evaluated on all *but* the training language. The results are a macro average over all the languages (in-language results) or all language pairs (cross-language results). For each model, language, and task we do 3 probe training runs with different random seeds. The error bars represent one standard deviation computed with bootstrapping by randomly sampling seeds for each language.

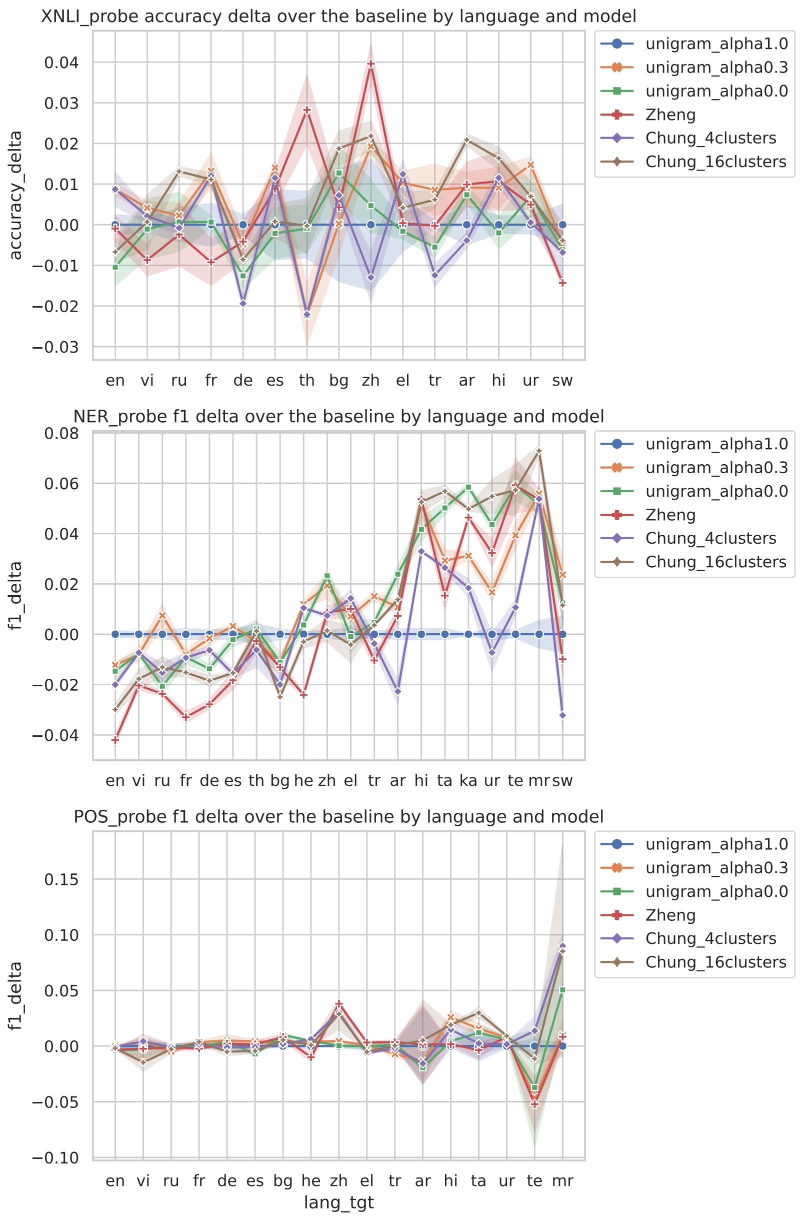


Figure 6.8 We zoom in on the in-language results from Figure 6.7a and compare the performance of the balanced tokenizers against the unbalanced Unigram tokenizer with $\alpha = 1.0$ over all tested languages for the tasks. In the case of the word-level tasks, especially in the case of named entity recognition, we observe a clear trend in line with our tokenizer investigations in 6.5. The balancing methods improve the language representations for word-level tasks. For the sentence-level tasks, we do not observe any systematic effects. The error bands are one standard deviation computed from the three probe training runs with different random seeds.

smaller magnitude, which is in line with our previous observations in Figure 6.5 in the intrinsic evaluation. In the case of POS, we do see more variance in results towards the low-resource languages but the effect is not as clear as for NER. We see significant improvements with Zheng and Chung methods on Chinese which correspond to the AR improvement for Chinese for the Zheng method in Figure 6.3 but we do not see any difference in Chinese tokenizer metrics in the case of Chung (Figure 6.5). We also see some improvements in Hindi, Tamil, and Marathi. For Telugu, we see a surprising drop for all methods except Chung with 4 clusters.

We turn our attention to the cross-language results in Figure 6.9. Here we again do not see any patterns in the NLI task, other than an overall drop in performance for Chung with 4 clusters, for which we do not have an explanation. On the other hand, we see clear patterns in NER and POS tasks. For both tasks, we see that the performance for low-resource languages is improved over the unbalanced baseline. Moreover, we see that the balancing has a net positive effect even for the high-resource languages, although the effect is not as pronounced as for the low-resource languages. In the NER task, we see that the Chung method with 4 clusters does not seem to improve on low-resource language as much as the other methods.

Direct comparison between intrinsic and extrinsic metrics

Lastly, we plot the differences in tokenizer metrics against the differences in downstream task scores in scatter matrices. This allows us to directly compare the relationship between intrinsic tokenizer metrics and downstream performance.

In Figure 6.10 we show the in-language results and in Figure 6.11 we show the cross-language results.

In the case of in-language results, we see that for the NER and POS tasks, there is a significant Spearman correlation between the differences in tokenizer metrics and the differences in task performance (0.84 and 0.34 Spearman correlation respectively). For the NLI task, we do not observe any significant correlations. This is in line with our observations from the language-level results (Figure 6.8).

In the case of cross-language results, we observe significant, low, negative Spearman correlations between JSD and word-level tasks NER and POS (-0.21 and -0.22 respectively). We also see significant correlations between combined CPT/AR and all downstream tasks (0.14 for NLI, 0.39 for NER, and 0.29 for POS). The negative correlation between JSD and word-level tasks is at odds with our finding in Chapter 4 and supports our hypothesis that the correlation between JSD and downstream performance is confounded by the vocabulary allocation. The positive correlation between CPT/AR and downstream performance is in line with our findings in Chapter 4 and suggests that the downstream performance

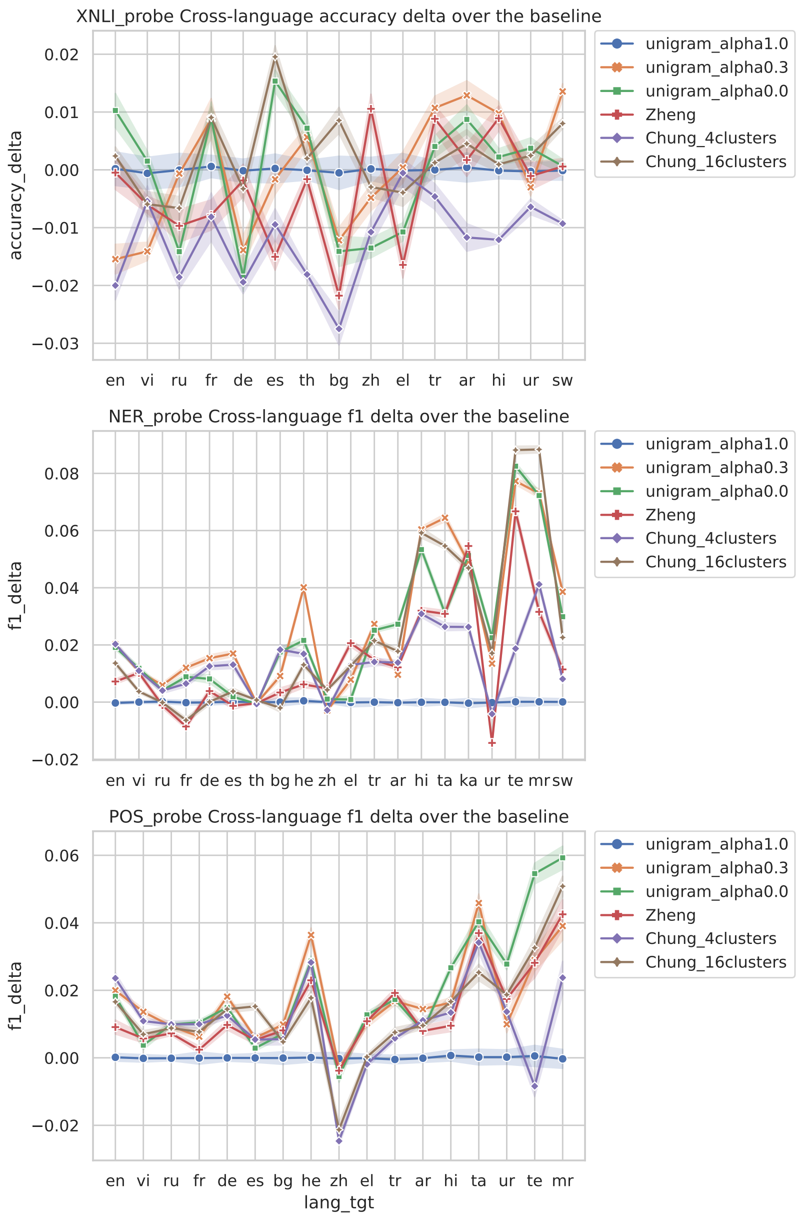


Figure 6.9 Here we investigate in detail the cross-lingual results from Figure 6.7b with comparison to the unbalanced Unigram tokenizer with $\alpha = 1.0$. We observe that word-level task transfers behave in line with the tokenizer investigations in 6.5. Moreover, it seems that both high-resource and low-resource languages benefit from the balancing methods, although the change is most clear on the low-resource side. For the sentence-level tasks, we do not observe any systematic effects. The value for each language and model is computed by averaging the difference in cross-lingual performance for the given *target* language over the baseline.

is positively influenced by the increase in token length in source and target languages. [2]

Conclusion of the extrinsic evaluation

Collecting all our observations together, we see that the tokenizers that aim to balance low-resource and high-resource languages do influence the results of downstream tasks compared to an unbalanced tokenizer. We see that the effect varies by task and language.

Generally, we see a high influence on word-level tasks (NER and POS) and no significant influence on NLI. The effect of balancing is most prominent for in-language NER and cross-language NER and POS. Interestingly, in cross-language, the balancing effect is a net positive even for the high-resource languages.

For all cases where the balancing effect is clear (NER in-language, NER/-POS cross-language), the best overall performance is achieved by the Unigram tokenizer trained on a balanced train set with $\alpha = 0.0$ and 0.3 (Figure 6.7).

6.3 Findings

We find that the balancing methods of Chung et al. [10], Zheng et al. [11], and Liang et al. [12] improve the representation of low-resource languages by increasing the *vocabulary allocation* for these languages at the cost of lowering the *vocabulary allocation* for the high-resource ones.

We also find that the Unigram tokenizer trained with a balanced dataset ($\alpha = 0.0$) achieves a similar effect as the replicated methods.

We find a striking similarity between the Unigram tokenizer trained on a balanced dataset and the Zheng method. We find that by maximizing the ALP across languages, Zheng method achieves a similar effect to running an Unigram tokenizer training on a balanced dataset.

We find that the clustering methods with a high number of clusters behave similarly to the Unigram tokenizer trained on a balanced dataset with exceptions of the low-resource languages, that happen to be clustered together with high-resource ones. We assume that the reason is that the clustering methods with a high k are similar to the Zheng method in that they train separate tokenizers for the majority of languages and only a few languages are grouped together.

[2] A possible explanation of the contradictory findings of the JSD influence may be observed in the summary plot Figure 6.2. We see that in our first batch of Huggingface experiments JSD correlates positively with AR, on the other hand the current batch of experiments shows a negative correlation.

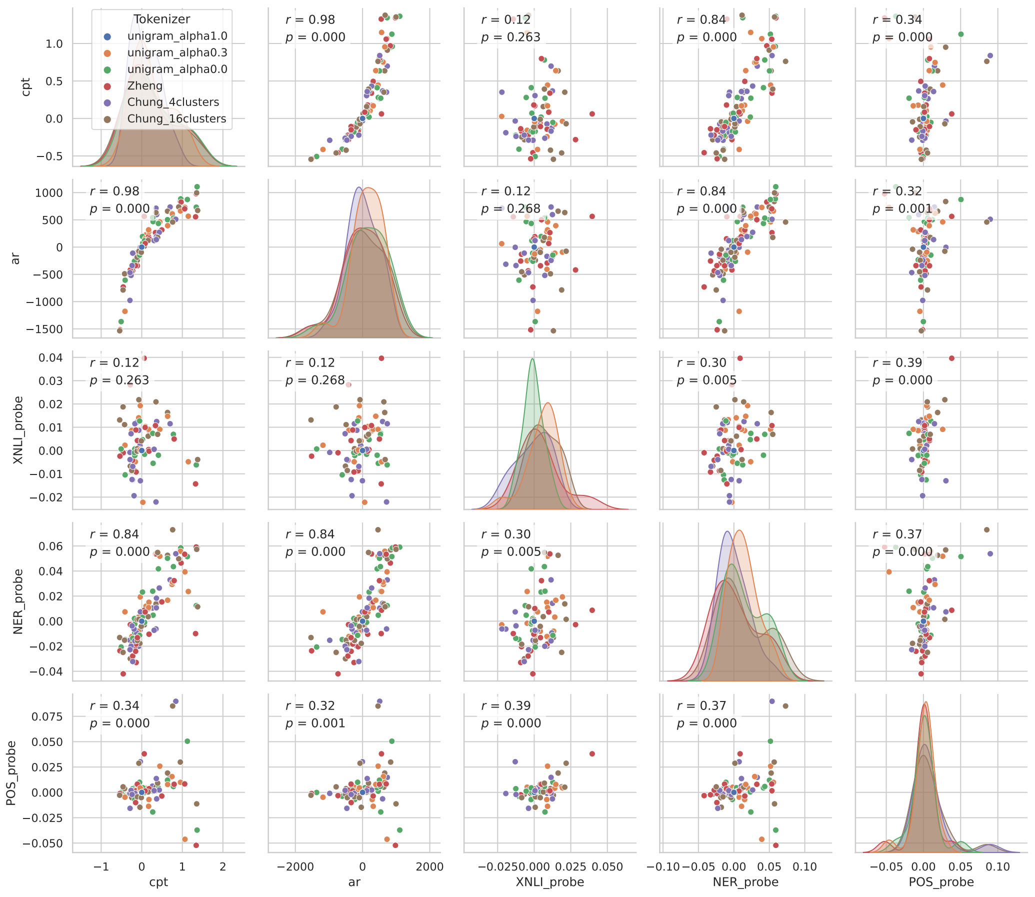


Figure 6.10 We visualize the in-language results from Figure 6.7a in a scatter matrix. We substract mean result across three tokenization mwethods from per-language results. We see significant Spearman correlations for the NER and POS tasks, although for POS the correlation is low. For the NLI task, we do not observe any significant correlations.

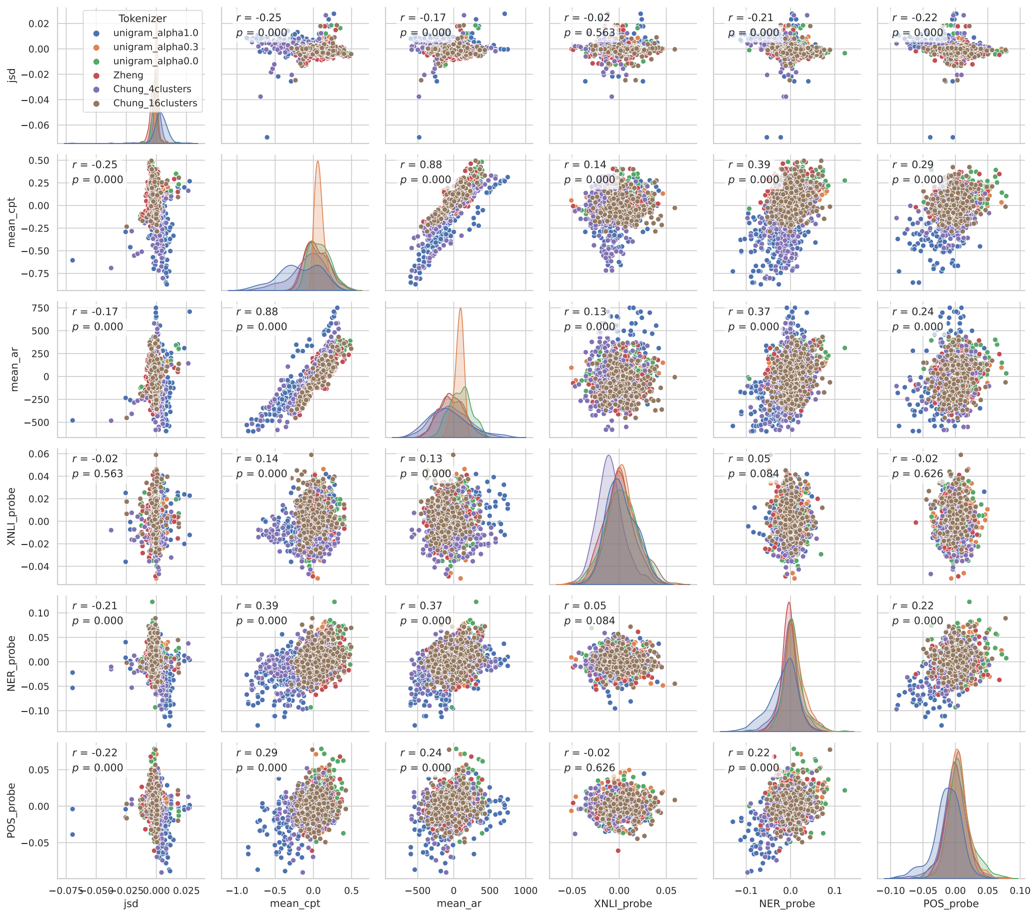


Figure 6.11 We visualize the cross-language results from Figure 6.7b in a scatter matrix. We substract mean performance across the three models from the language pairs results of vocabulary overlap metric (JSD) and the combined CPT/AR of the source and target languages. We see significant, low negative correlations between JSD and F1 scores for the NER and POS tasks and higher, significant correlations between combined CPT and F1 scores. This suggests that the word-level tasks benefit only slightly from an increase in overlap (decrease in JSD) and an increase in token length in source and target languages. For the NLI task, we observe a significant, low correlation between combined CPT/AR and NLI accuracy. This suggests that the cross-lingual transfer for sentence-level tasks benefits very slightly from an increase in allocation in source and target languages.

On the other hand, we find that the clustering methods with lower number of clusters yield more distinct results. Nevertheless, we observe that they are even more susceptible to a decrease in performance when high-resource and low-resource languages are assigned to the same cluster.

By running the extrinsic evaluation, we validate the observations we made using the intrinsic evaluation. We find that the balancing methods improve the word-level tasks for low-resource languages while having no impact on the sentence-level task.

Interestingly, we also find that the balancing has a net-positive effect for cross-lingual transfer across all languages.

Our findings suggest that in our scaled-down setting of 20 languages and 120k vocabulary size, the simpler method of training a standard Unigram tokenizer on a joint, balanced corpus is sufficient to create a good multilingual vocabulary that represents all languages well.

Chapter 7

Conclusion

In this thesis, our main focus was to explore the characteristics of tokenization methods for multilingual models and investigate their impact on language model representation quality.

To this end, we proposed a set of metrics for measuring the vocabulary allocation and overlap of tokenizer vocabularies. We compared our metrics to existing ones and we proposed a set of experiments to validate the usefulness of our metrics and investigate the properties of tokenization methods.

We found that our metrics are useful for assessing the differences between tokenization methods and that vocabulary allocation and overlap are good predictors of language model performance for word-level tasks. Especially when considering word-level downstream tasks, the learned contextualized representations are better when the tokenizer segments the text into longer tokens (high characters per token), when the tokenizer allocates tokens more uniformly (high average rank), and when there is less overlap between the languages (high Jensen-Shannon divergence).

With our established methodological framework, we then investigated the reasons for the subpar performance of standard tokenization methods in the multilingual setting. We demonstrated that to train a satisfactory tokenizer, we need around 100k to 1M lines of text per language. We also showed that the alphabet size does not largely affect our proposed metrics and that the default setting of covering 99.95% of Unicode characters leads to a well performing tokenizer.

We found that the main factors impacting the tokenization quality are the choice of implementation and data imbalance. We found that the Huggingface library yields subpar Unigram tokenizers and using the original Sentencepiece library mitigates a large gap in vocabulary allocation metrics we observed. More importantly, our research highlighted the strong impact of the data imbalance between high-resource and low-resource languages on the resulting tokenizer.

Our experiments indicate, that the standard tokenizer training method is susceptible to training data imbalance which leads to a decrease in tokenization quality for the low-resource languages. Sampling data uniformly from each language during the tokenizer training mitigates this effect and leads to a better tokenization for the underresourced languages at a smaller cost on the high-resource.

After investigating the standard tokenization scheme, we turned our attention to three existing works by Chung et al. [10], Zheng et al. [11], and Liang et al. [12] proposed for improving tokenization for low-resource languages. We identified that these vocabulary balancing methods use a highly unbalanced baseline to report their empirical improvements. We therefore investigated, how exactly these methods improve the tokenization quality for low-resource languages and how they compare to the standard method of training the tokenizer on a balanced and unbalanced joint corpus.

We found a surprising correspondence between the balancing methods and the Unigram tokenizer trained on a uniformly sampled dataset. We found that the balancing methods improve the tokenization quality for low-resource languages by increasing the vocabulary allocation for these languages at the cost of lowering the vocabulary allocation for the high-resource ones, similar to the standard method trained on uniformly sampled languages. Moreover, we found that the methods based on clustering are susceptible to a decrease in performance when high-resource and low-resource languages are assigned to the same cluster.

Our findings show that for all methods, the improvements on low-resource representation translate to improvement in word-level tasks in both in-language setting as well as cross-lingual transfer.

Overall, our findings contribute to understanding tokenization methods for multilingual models. We propose a methodology for assessing the differences between tokenizers, emphasize the importance of selecting appropriate implementation, training parameters and data balance, and we show that with proper care, the simpler tokenization method of training a tokenizer on joint, multilingual corpus can achieve similar results to the balancing methods in our experimental setting.

7.1 Limitations and future work

Lastly, we acknowledge that due to computational limitations, we were not able to compare tokenizers on a larger scale. The language models were scaled down, the training data subsampled, vocabulary size reduced, and the number of languages limited to 20. We believe that our methodology is a contribution in itself and following it leads to a better understanding of the reasons for the differences between tokenizers. On the other hand, we are aware that the findings of our experiments are limited to the experimental setting we used.

For future work, we would like to explore our research questions on a larger scale. Moreover, it would be interesting to investigate deeper the Huggingface BPE tokenizer as we found it to have the best overall performance in our experiments.

Bibliography

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423 (visited on 05/14/2023).

[2] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[3] Karthikeyan K et al. "Cross-Lingual Ability of Multilingual BERT: An Empirical Study". en. In: Feb. 2022. URL: https://openreview.net/forum?id=HJeT3yrtDr (visited on 03/30/2023).

[4] Alexis Conneau et al. "Unsupervised Cross-lingual Representation Learning at Scale". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747 (visited on 02/04/2023).

[5] Alexis Conneau et al. "Unsupervised Cross-lingual Representation Learning at Scale". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747 (visited on 02/04/2023).

[6] Kaj Bostrom and Greg Durrett. "Byte Pair Encoding is Suboptimal for Language Model Pretraining". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 4617–4624. DOI: 10.18653/v1/2020.findings-emnlp.414. URL: https://aclanthology.org/2020.findings-emnlp.414 (visited on 02/28/2023).

[7] Phillip Rust et al. “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)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 3118–3135. DOI: 10.18653/v1/2021.acl-long.243. URL: https://aclanthology.org/2021.acl-long.243 (visited on 03/01/2023).

[8] Taku Kudo and John Richardson. “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*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. DOI: 10.18653/v1/D18-2012. URL: https://aclanthology.org/D18-2012 (visited on 02/14/2023).

[9] Thamme Gowda and Jonathan May. “Finding the Optimal Vocabulary Size for Neural Machine Translation”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 3955–3964. DOI: 10.18653/v1/2020.findings-emnlp.352. URL: https://aclanthology.org/2020.findings-emnlp.352 (visited on 03/23/2023).

[10] Hyung Won Chung et al. “Improving Multilingual Models with Language-Clustered Vocabularies”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 4536–4546. DOI: 10.18653/v1/2020.emnlp-main.367. URL: https://aclanthology.org/2020.emnlp-main.367 (visited on 07/20/2023).

[11] Bo Zheng et al. “Allocating Large Vocabulary Capacity for Cross-Lingual Language Model Pre-Training”. en. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 3203–3215. DOI: 10.18653/v1/2021.emnlp-main.257. URL: https://aclanthology.org/2021.emnlp-main.257 (visited on 02/02/2023).

[12] Davis Liang et al. *XLM-V: Overcoming the Vocabulary Bottleneck in Multilingual Masked Language Models*. en. arXiv:2301.10472 [cs]. Jan. 2023. URL: http://arxiv.org/abs/2301.10472 (visited on 02/01/2023).

[13] Tomasz Limisiewicz, Jiří Balhar, and David Mareček. “Tokenization Impacts Multilingual Language Modeling: Assessing Vocabulary Allocation and Overlap Across Languages”. In: *Findings of the Association for Computational*

Linguistics: ACL 2023. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 5661–5681. URL: https://aclanthology.org/2023.findings-acl.350 (visited on 07/20/2023).

[14] Jacob Devlin. *bert/multilingual.md at master · google-research/bert*. en. July 2019. URL: https://github.com/google-research/bert/blob/master/multilingual.md (visited on 07/16/2023).

[15] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[16] Sebastian Ruder et al. “XTREME-R: Towards More Challenging and Nuanced Multilingual Evaluation”. en. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 10215–10245. DOI: 10.18653/v1/2021.emnlp-main.802. URL: https://aclanthology.org/2021.emnlp-main.802 (visited on 05/18/2023).

[17] Rico Sennrich, Barry Haddow, and Alexandra Birch. “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)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://aclanthology.org/P16-1162 (visited on 04/24/2023).

[18] Philip Gage. “A new algorithm for data compression”. In: *The C Users Journal* 12.2 (Feb. 1994), pp. 23–38. ISSN: 0898-9788.

[19] Mike Schuster and Kaisuke Nakajima. “Japanese and Korean voice search”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. Mar. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

[20] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[21] Taku Kudo. “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)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 66–75. DOI: 10.18653/v1/P18-1007. URL: https://aclanthology.org/P18-1007 (visited on 03/01/2023).

[22] Guillaume Lample and Alexis Conneau. *Cross-lingual Language Model Pretraining*. en. arXiv:1901.07291 [cs]. Jan. 2019. URL: http://arxiv.org/abs/1901.07291 (visited on 02/04/2023).

[23] Yinhan Liu et al. “Multilingual Denoising Pre-training for Neural Machine Translation”. In: *Transactions of the Association for Computational Linguistics* 8 (2020). Place: Cambridge, MA Publisher: MIT Press, pp. 726–742. DOI: 10.1162/tacl_a_00343. URL: https://aclanthology.org/2020.tacl-1.47 (visited on 05/14/2023).

[24] Linting Xue et al. “mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41 (visited on 05/14/2023).

[25] Teven Le Scao et al. “Bloom: A 176b-parameter open-access multilingual language model”. In: *arXiv preprint arXiv:2211.05100* (2022).

[26] Xi Victoria Lin et al. “Few-shot Learning with Multilingual Generative Language Models”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 9019–9052. URL: https://aclanthology.org/2022.emnlp-main.616 (visited on 07/20/2023).

[27] Junjie Hu et al. “XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalisation”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 4411–4421. URL: https://proceedings.mlr.press/v119/hu20b.html.

[28] Hai Wang et al. “Improving Pre-Trained Multilingual Model with Vocabulary Expansion”. In: *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 316–327. DOI: 10.18653/v1/

K19-1030. URL: https://aclanthology.org/K19-1030 (visited on 03/30/2023).

[29] Jonathan H. Clark et al. “Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation”. en. In: *Transactions of the Association for Computational Linguistics* 10 (Jan. 2022), pp. 73–91. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00448. URL: https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00448/109284/Canine-Pre-training-an-Efficient-Tokenization-Free (visited on 07/16/2023).

[30] Yi Tay et al. “Charformer: Fast Character Transformers via Gradient-based Subword Tokenization”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=JtBRnrlOEFN.

[31] Linting Xue et al. “ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models”. In: *Transactions of the Association for Computational Linguistics* 10 (2022). Place: Cambridge, MA Publisher: MIT Press, pp. 291–306. DOI: 10.1162/tacl_a_00461. URL: https://aclanthology.org/2022.tacl-1.17 (visited on 07/20/2023).

[32] Sabrina J. Mielke et al. “Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP”. In: *ArXiv* (Dec. 2021). URL: https://www.semanticscholar.org/paper/Between-words-and-characters%3A-A-Brief-History-of-in-Mielke-Alyafeai/d617f51833860dc50d202af7f80be71304b2e994 (visited on 02/05/2023).

[33] Phillip Rust et al. “Language Modelling with Pixels”. In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=FkSp8VW8RjH.

[34] Elizabeth Salesky, David Etter, and Matt Post. “Robust Open-Vocabulary Translation from Visual Text Representations”. en. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 7235–7252. DOI: 10.18653/v1/2021.emnlp-main.576. URL: https://aclanthology.org/2021.emnlp-main.576 (visited on 07/16/2023).

[35] Elman Mansimov et al. “Towards End-to-End In-Image Neural Machine Translation”. en. In: *Proceedings of the First International Workshop on Natural Language Processing Beyond Text*. Online: Association for Computational Linguistics, 2020, pp. 70–74. DOI: 10.18653/v1/2020.nlpbt-1.8. URL: https://www.aclweb.org/anthology/2020.nlpbt-1.8 (visited on 07/16/2023).

[36] Guillaume Wenzek et al. “CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data”. English. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 4003–4012. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.494 (visited on 07/20/2023).

[37] Vilém Zouhar et al. “Tokenization and the Noiseless Channel”. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 5184–5207. URL: https://aclanthology.org/2023.acl-long.284 (visited on 07/20/2023).

[38] Shijie Wu and Mark Dredze. “Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT”. 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)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 833–844. DOI: 10.18653/v1/D19-1077. URL: https://aclanthology.org/D19-1077 (visited on 07/01/2023).

[39] Alexis Conneau et al. “What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 2126–2136. DOI: 10.18653/v1/P18-1198. URL: https://aclanthology.org/P18-1198 (visited on 07/03/2023).

[40] Yonatan Belinkov, Sebastian Gehrmann, and Ellie Pavlick. “Interpretability and Analysis in Neural NLP”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*. Online: Association for Computational Linguistics, July 2020, pp. 1–5. DOI: 10.18653/v1/2020.acl-tutorials.1. URL: https://aclanthology.org/2020.acl-tutorials.1 (visited on 07/03/2023).

[41] Terra Blevins, Hila Gonen, and Luke Zettlemoyer. “Analyzing the Mono- and Cross-Lingual Pretraining Dynamics of Multilingual Language Models”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 3575–3590. URL: https://aclanthology.org/2022.emnlp-main.234 (visited on 07/03/2023).

[42] Joakim Nivre et al. “Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection”. English. In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language

Resources Association, May 2020, pp. 4034–4043. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.497 (visited on 07/20/2023).

[43] Xiaoman Pan et al. “Cross-lingual Name Tagging and Linking for 282 Languages”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1946–1958. DOI: 10.18653/v1/P17-1178. URL: https://aclanthology.org/P17-1178 (visited on 07/03/2023).

[44] Afshin Rahimi, Yuan Li, and Trevor Cohn. “Massively Multilingual Transfer for NER”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 151–164. DOI: 10.18653/v1/P19-1015. URL: https://aclanthology.org/P19-1015 (visited on 07/03/2023).

[45] Alexis Conneau et al. “XNLI: Evaluating Cross-lingual Sentence Representations”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2475–2485. DOI: 10.18653/v1/D18-1269. URL: https://aclanthology.org/D18-1269 (visited on 02/04/2023).

[46] Mikel Artetxe and Holger Schwenk. “Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond”. In: *Transactions of the Association for Computational Linguistics* 7 (2019). Place: Cambridge, MA Publisher: MIT Press, pp. 597–610. DOI: 10.1162/tacl_a_00288. URL: https://aclanthology.org/Q19-1038 (visited on 07/03/2023).

[47] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955). Publisher: Wiley Online Library, pp. 83–97.

[48] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

Appendix A

Sentencepiece default parameters

Parameter Name	**Explanation**	**Default**
model_type	Model algorithm: unigram, bpe, word, or char	”unigram”
vocab_size	Vocabulary size	8000
character_coverage	Character coverage to determine the minimum symbols	0.9995
input_sentence_size	Maximum size of sentences the trainer loads	0
seed_sentencepiece_size	The size of seed sentencepieces	1000000
shrinking_factor	Keeps top shrinking_factor pieces with respect to the loss	0.75
num_sub_iterations	Number of EM sub-iterations	2
max_sentencepiece_length	Maximum length of sentence piece	16
max_sentence_length	Maximum length of sentence in byte	4192

Table A.1 Default parameters for Sentencepiece training. We bold the parameters that we modify in some of our experiments. Summarized from `https://github.com/google/sentencepiece/blob/master/doc/options.md`