Ricardo Lukas Jung
6227492
Empirische Sprachwissenschaft (B.A.)
Phonetik & Digital Humanities
15th Semester
s2458588@stud.uni-frankfurt.de

Bachelor Thesis

Lexicalizing a BERT Tokenizer

Building Open-End MLM for Morpho-Syntactically Similar Languages

Ricardo Lukas Jung

Date of Submission:
February 12, 2023

Text Technology Lab
Prof. Dr. Alexander Mehler
Dr. Zakharia Pourtskhvanidze

Erklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

Ort, Datum

Unterschrift

Abstract

This is the abstract: what is this about? what was done? what where the results?

Contents

List of Figures

List of Tables

List of Acronyms

bbgc bert-base-german-cased

BERT Bidirectional Encoders from Transformers

BPE Byte Pair Encoding

BWE Bert WordPiece

CL Computational Linguistics

GerParCor German Parliamentary Corpus

HanTa Hanover Tagger

LM Language Model

LSTM Long Short-Term Memory

ML Machine Learning

MLM Masked Language Model

NLP Natural Language Processing

POS Part of Speech

TTLab Text Technology Lab

WM Wordmap

arabic

1 Introduction

2 Overview

3 Methodology

biblatex booktabsin this section the whole methodoloy is covered. what do i use in this thesis, why do i use it and lastly, how? make sure the why covers methodological implications. (vergiss nicht alle pakete als quelle im Anhang)

3.1 Requirements

A series of tools will help to achieve lexicalized tokenization. They will be explained in this chapter along with their methodological edge.

3.1.1 Pipeline Structure & Transformers Library

Bidirectional Encoders from Transformers (BERT) is a language learning transformer model designed for Natural Language Processing (NLP) tasks (Vaswani et al. 2017). Upon release it achieved higher performance scores compared to previously used Long Short-Term Memory (LSTM) models (Devlin et al. 2018). Two main model characteristics can be observed for BERT. Firstly, it is the first Language Model (LM) to implement simultaneous attention heads, allowing for bidirectional reading. The methodological implication of reading to the left and right of a token is to include more information about the language in single embeddings. Secondly, BERT introduced the (at the time novel) Masked Language Model (MLM) method for training. The method involves masking a specified amount (default 15%) of random tokens in the input sequence. Masked tokens are guessed by the model which can then update its weights according to success or failure.

The NLP community has since developed BERT and adapted it to the needs of contemporary NLP problems (roberta, germanbert, mbert CITATION). Its wide support, comparability and versatility make BERT the model of choice for this thesis. Another notable feature in BERT is the implementation of the WordPiece tokenizer module (QUELLE?). Default BERT WordPiece tokenization is predominantly heuristic by combining strings based on a precalculated score. A variety of pre-trained tokenizers are available, although they come with a caveat. Once a tokenizer is trained on a dataset it is specific to that dataset. This means the application of a tokenizer on another dataset may result in out-of-vocabulary issues and

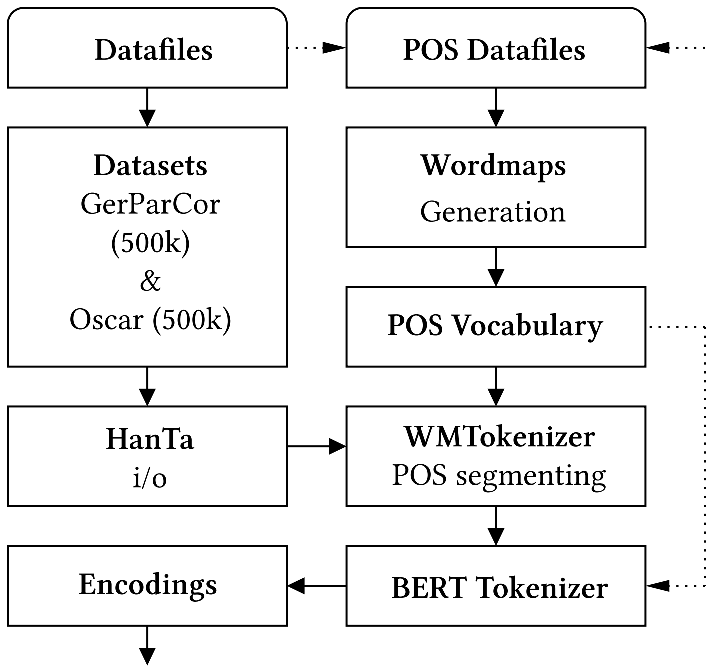

Figure 1: Schematic integration of Wordmap segmentation into the BERT architecture. Continuous lines show non-selective channels, while dotted lines only pass data selectively

different token/subtoken distributions.

Particularly relevant to this thesis is the option to train an own tokenizer from the base module. Usually, WordPiece generates its own set of subtokens called *vocabulary*, which is used by the standard tokenizer to generate unique input numerical IDs. These identifiers correspond to tokens or parts of tokens found in the original dataset that the model was trained on. Wordmap partly takes over the tokenization depending on the the type of input it receives, the exact process is given in subsection 3.2.1. Once a string is tokenized it gets passed on to the transformer model for contextualized embedding. Schematic Figure 1 shows the processing of data up until token encoding.

In Figure 1, Datafiles contains the raw unprocessed data coming from corpora. The input is provided in txt-files holding sentences line by line. Two datasets are generated from this input, each amounting to 500000 sentences per corpus for fine-tuning. POS Datafiles ideally contains only words that are identified as the same Part of Speech (POS) category. To increase the resulting POS vocabulary, POS Datafiles can be augmented by two channels: input can come from the corpus itself, but would need to be tagged first. Otherwise a predefined list of words can be used as an external and generic source for Wordmap generation. In POS Vocabulary, naive adjustments like removing outliers and interferences are performed on the set of subwords yielded by Wordmap generation. It is important to note that the POS vocabulary is separate from the the tokenizer vocabulary. This serves the specific purpose of keeping the vocabulary on which the model is trained clean from unwanted or unused subwords. Machine learning practice generally points to the trade-off between vocabulary size and model performance, hence the addition of POS Vocabulary to the BERT Tokenizer vocabulary is used only when necessary. Datasets are then passed through Hanover Tagger (HanTa) to provide WMTokenizer with flagged data to tokenize only selected tokens. Lastly BERT Tokenizer returns the canonical encodings known from language modeling.

3.1.2 Data

Two corpora where selected for fine-tuning: German Parliamentary Corpus (GerParCor) and oscar (OSCAR 2022). FLOTA was trained on 12000 samples per category Hofmann, Schuetze, and Pierrehumbert 2022, which is why for this fine-tuning a sample size of 500000 seems sufficient. GerParCor is a "GerParCor genre-specific corpus of (predominantly historical) German-language parliamentary protocols from three centuries and four countries, including state and federal level data." (Abrami et al. 2022, p. 1). Of all subcorpora included in GerParCor, one specifically texts and transcripts of the german parliament. All 500k samples in the GerParCor dataset used for this thesis is found in the `Bundestag` subcorpus of GerParCor. Oscar offers other subcorpora partly hosted on huggingface. In this case, a web-crawled german corpus called `unshuffled_deduplicated_de` was chosen, as introduced by its curators Ortiz Su'arez, Sagot, and Romary (2019); (2020).

The POS vocabulary comes from Wiktionary 2022 hanta-crawled verbs of gerparcor Wartena 2019

3.1.3 Benchmark

The benchmark used on the model is oLMpics (Talmor et al. 2020)

3.2 Implementation

Tatsächliche Anwendung der Methoden auf die Daten

3.2.1 Tokenizer

ESSENTIALLY DERIVING SENSIBLE SUBTOKENS TO REPRESENT LEXEMES The methods of tokenization is explained in the following. This is the main part of the thesis containing two algorithms for vocabulary generation and token segmentation. Once the two algorithms have been explained, the last section of this chapter features the

Languages, as consistent form of communication, always Throughout this section, the words $target$ and $token$ are used to describe a word that is analyzed. One denotes the argument of the wordmapping and segmenter function ($target$), the other describes a word occurring in the corpus ($token$).

Generating a custom pre-training vocabulary

Embedding those subwords which take part in inflectional processes essentially means deriving sensible subwords to represent actual morphemes.

The vocabulary generated for this chapter is a list of inflected and non-inflected verbs (as to state the example) provided by two sources: (1) the german verb wiktionary and (2) a crawl of the GerParCor corpus with the HanTa tagger Wartena 2019. Initial experiments where done only with the wiktionary verb list since it provided sanitized input for the algorithm in development. As soon as the verbs where extracted from GerParCor via HanTa

The Wordmap algorithm as shown in Algorithm 1 is the first step to extracting morphemes from a token. Its purpose is to compare two strings and store their intersections in a map of boolean values.

Wordmap requires two **inputs** $verbs$ and $target$. The resulting wordmap will be generated for $target$, while $verbs$ serves as comparison. Any map generated from this also has the same length as $target$. $verbs$ is a set of tokens pertaining to the same POS category. Note that $verbs$ should only contain those POS-tagged tokens that are expected to carry lexical information (e.g. verbs, adjectives, etc.). The set is previously extracted from the corpus by POS-tagging. Optionally, the set can be augmented by manually adding POS matching tokens from external sources. The 2-tuple $pair$ are the strings to be compared. It is passed on to (1) SHORTER, a function returning the shorter of both strings (2) LONGER, expectedly returning the longer of both strings (3) MATCH_CASE, a function to determine the behavior of the algorithm later on. As two strings are compared MATCH_CASE captures three cases: $pair$ matches in the first, last or both positions. Finally len denotes the length of the longest string and δ difference in length between s and l. δ functions as an offset for index-based comparisons WORDMAP.

WORDMAP is a naive mapping function generating the wordmaps.

Once every v has been compared to $target$, $maps$ stores boolean counts of characters occurring in their respective positions in $target$. Every map is cleaned with a regular expression to reduce noise caused by natural character occurrence (some characters like <n>,

Algorithm 1 Wordmap generation

Input: $verbs = \{v : POS\}, target$ ▷ *verbs*: set of single-POS lexemic tokens

Output: $maps = (map_1, \ldots, map_{|verbs|})$

function WORDMAP(w1, w2, d=0)
    $f_1 : c1, c2 \mapsto c1 == c2$
    $f_2 : \left(\sum_{i=0+d}^{|w1|} w1[i], w2[i] \mapsto f_1(w1[i], w2[i]) \right)$
    return f_2
end function

for $v \in verbs$ **do**
    $pair = (target, v)$
    $s =$ SHORTER$(pair)$
    $l =$ LONGER$(pair)$
    $case =$ MATCH_ENDS$(pair)$ ▷ Returns if strings match in the last or first position
    $len =$ LEN(l)
    $\delta = \Delta(len,$ LEN$(s))$

    if *case: any match* **then**
        if δ **then**
            if *case: left match* **then**
                WORDMAP(l, s)
                Pad map from right side with 0s to match len
            end if
            if *case: right match* **then**
                WORDMAP(l, s, δ)
                Pad map from left side with 0s to match len
            end if
        else
            WORDMAP(l, s)
        end if
    end if
end for

Table 1: Example wordmaps for $target$ = `verstehen`

String	Wordmap	Case	Padding
`verarbeiten`	111000000	left match	Yes
	001000011	right match	Yes
`variiert`	101001000	left match	Yes
`vormachen`	101000111	left match	No
	101000111	right match	No
`anstrebtest`		no match	
(...)			

<s> will be more frequent than others). Continuous concatenations of leading or trailing matches stay, while every match enclosed by 0 will be replaced the 0. As an example, $wordmap = 11101100101$ contains three matches on the inside which will result in 11100000001 as final output. In the penultimate stage of mapping $target$, all maps are summed up to receive the number of absolute positional occurrences of every character in $target$. This positional mapping allows for detecting relevant segments in a token based on a threshold. Characters in range of the predefined threshold are selected for the mapping of a target token. Functional morphemes (morphemes that are carriers of grammatical features) are typically much more frequent than their lexical counterparts. Consequently, lexical morphemes in the family of inflectional languages are - by definition - modified by functional morphemes, they occur much less frequently. In this case, the activation function for a concatenation of same boolean values to be selected as segment is the normalizing z-score function defined as: $z_i = \frac{x_i - \overline{x}}{S}$, where z is the z-score, $\overline{x}$ and S are the sample mean and the sample standard deviation.

BERT Tokenizer Modification

Algorithm 2 recursively computes every possible segmentation for a string $target$ from a given vocabulary pos_vocab from left to right. The vocabulary contains all the segments that were identified in the previous step by Algorithm 1. Every segmentation has to be complete so that its segments corresponds to non overlapping substrings of $target$. For every $target$ a subvocabulary $morpheme$ is defined, containing all strings that are in the vocabulary of **POS! (POS!)** members. This task is a weighted coverage problem in the NP-hard domain. Unary morphemes to the left are excluded from the pre-selected subvocabulary to drastically reduce the number of possible permutations , as they can be embedded in n-ary tokens as well. The recursion can be seen as n-ary a trees containing every permutation of the set

Algorithm 2 Target Segmentation

Input: $target, pos\_vocab$ ▷ Vocabulary consisting of verbs only

Output: $\{(tuples\ of\ subwords)\} \approx \{t \in \mathcal{P}(s \in pos\_vocab : s \in target) : t \equiv target\}$

```
segmentations = ()

function SEGMENTER(token, stop, start=0, segments)
    if start = stop then
        Add segments to segmentations
    else
        morphemes = (m ∈ pos_vocab : target.STARTSWITH(m) ∧ |m| > 1)
        for m ∈ morphemes do
            start += LEN(m)
            Add m to segments
            rest = target[LEN(m) :]
            if LEN(rest) == 1 then
                start = stop
                Add rest to segments
                Add segments to segmentations
                Decrement start, crop segments
            else
                SEGMENTER(target = rest, stop, start = start, segments)
                Decrement start, crop segments
            end if
        end for
    end if
end function
MAXIMIZE_SEGMENTS(segmentations)
```

Call:	SEGMENTER("verstehen")
Subvocab:	{versteh vers ver ve en teh steh ste st rsteh rst rste ehe he n}

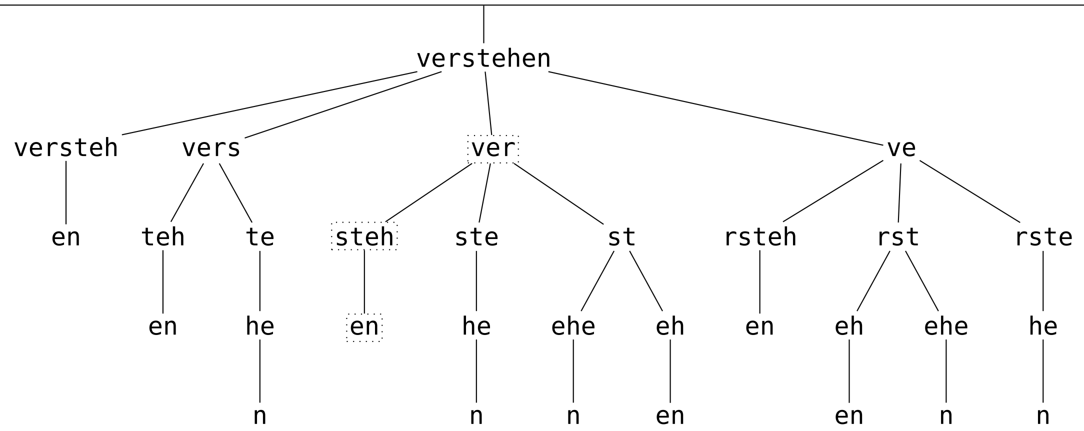

Figure 2: Permutations SEGMENTER generates from target i.e. `verstehen`. Dotted segments mark the segmentation chosen later by the maximization function during tokenization.

$morphemes$ where the sum of branches all satisfy $target$.

As shown in Figure 2, all morphemes that $target$ starts with are stored to form the first nodes of the permutation tree. Each time a morpheme is selected the index $start$ is incremented by the length of the morpheme to indicate when the string has been completely segmented. The new recursion is called with the updated index $start$ and $target$ sliced by the length of the morpheme contained in the parent node. Incomplete segmentations that miss exactly one character to the right are accepted with the added missing string. If the vocabulary cannot satisfy a segmentation because it is missing the necessary strings, segmentation is omitted and the original input token is returned as such.

Then, of all $segmentations$ a single segmentation is selected by a maximization function MAXIMIZE_SEGMENTS calculating weights for every segment. The maximization function for one segment is the defined as:

$$\underset{s\in S}{\arg\max} f(s) := \left\{ s \in S : \sum_{i=1}^{|s|} \sqrt[s_i]{\frac{s_i}{t} \div |s|} \right\} \tag{3.1}$$

Where S is a set of segmentation tuples s, $|s|$ is the length of the tuple (read: number of segments) and s_i indicates the length of the segment at position i in tuple s. For every segment (vertex) in a segmentation tuple (branch) the segment's length is divided by the token length t to get the coverage the morpheme provides towards the target string. The number of segments $|s|$ is a divisor meant to cap the number of segments in a segmentation. It prevents choosing a segmentation overflowing with too many short segments. Short segments are convenient for completing a segmentation, but will increase the chance of slicing a token where it linguistically lacks sense to do so. Lastly the ${s_i}^{\text{th}}$ root implements a bias against segments that are too long. While the accuracy decreases for longer words, this method of maximization performs reasonably well in the range of 1–4 syllable tokens, which make up around 87% of the verb vocabulary.

Ultimately, segmentation is a matter of interpretation. If there are several possible interpretations by wich to segment a word this tokenization method relies on the assumption that the comparison with POS- members displays the probable shape of segmentation for a token. As mentioned in 3.2.2, the default WordPiece Tokenizer lacks A linguistically informed

The field of NLP (Glück and Rödel 2016) has been expanded ever since the emergence of the language models. Natural language processing is understood as the

3.2.2 Model Training

All fine-tuned models use the bert-base-german-cased (bbgc) baseline by Chan, Schweter, and Möller (2019). For reference and readability they are given shortened IDs as follows:

ID	Full name
mwg5	mlm_wmt_gpc500k
mwo5	mlm_wmt_oscar500k
msg5	mlm_std_gpc500k
mso5	mlm_std_oscar500k
bbgc	bert-base-german-cased

Table 2: ID references assigned to full model names

The full model names in Table 2 captures broad characteristics of the model. For example, $\boxed{\text{mwo5}}$ is the $\boxed{\text{m}}$asked language model using $\boxed{\text{W}}$ordmap tokenization fine-tuned on the $\boxed{\text{O}}$scar dataset of $\boxed{\text{5}}$00k samples, while msg5 is the the masked language model using standard BERT wordpiece tokenization fine-tuned on the GerParCor dataset of 500k samples. All models except for bbgc are trained on the same parameters as seen in table Table 3.

The first four models use bbgc as baseline, meaning that they inherit its structural properties. DeepsetAI released bbgc having trained in two phases with differing sequence lengths

ID	Corpus	Tokenization	LR	Steps	Batchsize	WU	Base
mwg5	GerParCor	BWP + WM	0.0003	31250	16	500	bbgc
mwo5	Oscar	BWP + WM	0.0003	31250	16	500	bbgc
msg5	Gerparcor	BWP	0.0003	31250	16	500	bbgc
mso5	Oscar	BWP	0.0003	31250	16	500	bbgc
bbgc	Mixed	BWP	0.0001	810k/30k	1024	10000	-

Table 3: List of all used models and their hyperparameters. LR = learning rate, WU = warmup steps.

128/512 respectively. The maximum sequence length for each model is 512, the base vocabulary size is 30k. Models mwg5, mwo5, msg5, and mso5 where trained with less steps and thus higher learning rate to converge quicker with the given datasets. The models where implemented in PyTorch Paszke et al. (2019) and on the GPUs Quadro RTX 8000 (48Gb) and NVIDIA GeForce GTX 1080 Ti (11Gb) provided at Text Technology Lab (TTLab).

3.2.3 Benchmark

The original draft for this thesis featured a benchmark with a translated multiple choice question answering task from the oLMpics benchmark (Talmor et al. (2020)). Due to incompatible versioning dependencies this was not feasible and a substitute had to be found.

4 Results

4.1 Benchmark

This section is a comparative description of the results from the sequence classification task described in subsection 3.2.3. For every benchmark of three epochs precision, recall and F1 are given. Precision reflects the amount of correct predictions a model has made for that particular class. Recall shows how often the model correctly predicts a class in relation to all positive predictions. F1 is a score combining precision and recall is the *harmonic mean* of precision and recall, measuring the models accuracy.

There is a common trend in all runs displaying a growth in precision, recall and F1. This means that all models have continously improved predicting the class of a news title. The highest F1 in the last epoch achieved for any of the smaller models (mso5, msg5, mwo5, mwg5) is mwo5 with a score of 0.69194, meanwhile the lowest F1 is found in Table 5 at 0.590542 mwg5. Similar arrangement is found when comparing final test scores in the set of smaller models: mwo5 > mso5 > msg5 > mwg5 ($0.442827 > 0.405168 > 0.392490 > 0.389116$).

mwo5	**Epoch 1**	**Epoch 2**	**Epoch 3**	**Test score**
Precision	0.292614	0.446338	0.71387	0.449735
Recall	0.329531	0.552598	0.73384	0.474525
F1	0.242739	0.473851	0.69194	0.442827

Table 4: Metrics for masked language model trained on the Oscar dataset with Wordmap infused tokenization. Evaluated on sequence classification task.

mwg5	**Epoch 1**	**Epoch 2**	**Epoch 3**	**Test score**
Precision	0.237664	0.399781	0.603534	0.441891
Recall	0.244613	0.463878	0.637516	0.440304
F1	0.163024	0.389905	0.590542	0.389116

Table 5: Metrics for masked language model trained on the GerParCor dataset with Wordmap infused tokenization. Evaluated on sequence classification task.

mso5	Epoch 1	Epoch 2	Epoch 3	Test scores
Precision	0.269615	0.422096	0.596987	0.395879
Recall	0.351077	0.501901	0.657795	0.446388
F1	0.266260	0.412598	0.604824	0.405168

Table 6: Metrics for masked language model trained on the GerParCor dataset with bbgc tokenization. Evaluated on sequence classification task.

msg5	Epoch 1	Epoch 2	Epoch 3	Test scores
Precision	0.297466	0.517110	0.656808	0.439873
Recall	0.359949	0.544994	0.676806	0.439924
F1	0.267111	0.480420	0.626593	0.392490

Table 7: Metrics for masked language model trained on the GerParCor dataset with bbgc tokenization. Evaluated on sequence classification task.

4.2 Tokenization

Show specific examples of tokenization and analyze the qualitatively (maybe quantitatively)

bbgc	**Epoch 1**	**Epoch 2**	**Epoch 3**	**Test scores**
Precision	0.646150	0.768675	0.860180	0.622436
Recall	0.709759	0.804816	0.883397	0.637262
F1	0.660588	0.778371	0.868166	0.624789

Table 8: Metrics for masked language model baseline bert-base-german-cased[1]. Evaluated on sequence classification task.

Summary	**bbgc**	**std+oscar**	**std+gpc**	**wmt+oscar**	**wmt+gpc**
Precision	**0.622436**	0.395879	0.439873	0.441891	0.449735
Recall	**0.637262**	0.446388	0.439924	0.440304	0.474525
F1	**0.624789**	0.405168	0.392490	0.389116	0.442827

Table 9: Test score summary for all evaluated models.

5 Discussion

6 Conclusion

7 Testchapter

7.1 Citing

Abrami et al. 2022

7.2 Quoting

> “This is a quote by textquote” (DeepL 2021) “This is a quote by enquote”

7.3 Referencing

Short reference **??**
Long reference **??**
monofont for code or string `monofont`

Bibliography

Abrami, Giuseppe, Mevlüt Bagci, Leon Hammerla, and Alexander Mehler (June 2022). "German Parliamentary Corpus (GerParCor)". In: *Proceedings of the Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, pp. 1900–1906. URL: https://aclanthology.org/2022.lrec-1.202.

Chan, Branden, Stefan Schweter, and Timo Möller (June 2019). *GermanBERT*. URL: https://huggingface.co/bert-base-german-cased.

DeepL (Nov. 2021). *How does deepl work?* URL: https://www.deepl.com/en/blog/how-does-deepl-work. Last accessed: 28.12.2022.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

Glück, Helmut and Michael Rödel, eds. (2016). *Metzler Lexikon Sprache*. ger. 5th ed. Springer eBook Collection. Stuttgart: J.B. Metzler, pp. 141–142. ISBN: 978-3-476-05486-9. DOI: 10.1007/978-3-476-05486-9. URL: http://dx.doi.org/10.1007/978-3-476-05486-9.

Hofmann, Valentin, Hinrich Schuetze, and Janet Pierrehumbert (May 2022). "An Embarrassingly Simple Method to Mitigate Undesirable Properties of Pretrained Language Model Tokenizers". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Dublin, Ireland: Association for Computational Linguistics, pp. 385–393. DOI: 10.18653/v1/2022.acl-short.43. URL: https://aclanthology.org/2022.acl-short.43.

Ortiz Su'arez, Pedro Javier, Laurent Romary, and Benoit Sagot (June 2020). "A Monolingual Approach to Contextualized Word Embeddings for Mid-Resource Languages". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 1703–1714. URL: https://www.aclweb.org/anthology/2020.acl-main.156.

Ortiz Su'arez, Pedro Javier, Benoit Sagot, and Laurent Romary (2019). "Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures". en. In: ed. by Piotr Bański et al. Proceedings of the Workshop on Challenges in the Management of Large Corpora (CMLC-7) 2019. Cardiff, 22nd July 2019. Mannheim: Leibniz-Institut f"ur Deutsche

Sprache, pp. 9–16. DOI: 10.14618/ids-pub-9021. URL: http://nbn-resolving.de/urn:nbn:de:bsz:mh39-90215.

OSCAR (Jan. 2022). *The OSCAR project (Open Super-large Crawled Aggregated coRpus).* URL: https://oscar-project.org/.

Paszke, Adam et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* DOI: 10.48550/ARXIV.1912.01703. URL: https://arxiv.org/abs/1912.01703.

Talmor, Alon, Yanai Elazar, Yoav Goldberg, and Jonathan Berant (Dec. 2020). "oLMpics-On What Language Model Pre-training Captures". In: *Transactions of the Association for Computational Linguistics* 8, pp. 743–758. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00342. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00342/1923716/tacl_a_00342.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00342.

Vaswani, Ashish et al. (2017). *Attention Is All You Need.* DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Wartena, Christian (2019). "A Probabilistic Morphology Model for German Lemmatization". en. In: Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019), pp. 40–49. DOI: 10.25968/opus-1527. URL: http://nbn-resolving.de/urn:nbn:de:bsz:960-opus4-15271.

Wiktionary (Sept. 2022). *Verzeichnis: Deutsch.* URL: https://de.wiktionary.org/wiki/Verzeichnis:Deutsch.