Bad Pair Encoding: Language-specific Tokenization with a Pretrained LM CSE 481N

Kai Nylund, Alessio Tosolini, and Elliott Zackrone

Spring 2024

Abstract

Byte Pair Encoding (BPE) has been the de facto tokenization strategy since the dawn of transformers in the late 2010s, but does not take into consideration underlying linguistic structure. In multilingual models, BPE frequently results in oversegmentation of low-resource languages, and languages not written in the Latin script. Pretraining a model from scratch using a low-resource language specific tokenizer, however, tends to be prohibitively expensive or impossible due to a lack of data. To remedy this issue, we evaluate the efficacy of finetuning models on translation tasks for German, Russian, and Finnish using tokenizers different from their pretraining. We finetune mT5 with two alternative tokenization strategies: (i) first tokenizing with a BPE trained on datasets containing only English and the target language and (ii) using a morpheme tokenizer that greedily attempts to split every unknown word into its best guess morphemes. Although these tokenization strategies reduce loss when finetuning on English translation to the target language, we find that they both induce oversegmentation and harm downstream performance. Future work will either need to pretrain from scratch or modify a model's embeddings to evaluate the effect of linguistically-informed tokenization. We publicly release our code at: `https://github.com/KaiNylund/bad-pair-encoding`.

1 Introduction and Motivation

Being the first step in most natural language processing systems, tokenization plays an integral role in the success of a model. Although there has been much research on find good tokenization strategies for various NLP tasks, Byte Pair Encoding (BPE) (Gage [1994]) tends to be the most common tokenization system since the mid-2010s. However, although BPE's language-independence is useful for generalization to multilingual language modeling, it is not linguistically informed (i.e. it does not explicitly aim to understand the linguistic structures). This flaw becomes most evident with specific downstream tasks where linguistic information is more valuable, such as machine translation. For example, linguistically informed tokenization has demonstrated improvement for English-Korean translation tasks (Park et al. [2020]). However, pretraining models from scratch can be computationally expensive and often infeasible. Our project aims to determine if finetuning existing language models using a linguistically-informed tokenizer can increase accuracy on a translation task. This question is especially important for low-resource languages, where finetuning a preexisting multilingual model results in better accuracy than training a model from scratch due to data scarcity.

We explore this question in two ways. First, we train tokenizers on datasets composed of the source and target languages, attempting to remedy the fact that general-purpose tokenization in pretrained multilingual models often marginalizes low-resource languages (Ahia et al. [2023]). Second, we tokenize on morpheme

boundaries using a greedy morpheme tokenizer, attempting to create a linguistically informed tokenization system that increases accuracy by better modelling the linguistic structures of words. In other words, our project explores whether we can improve translation by only changing the tokenization.

2 Technical Idea Section(s)

2.1 What's wrong with BPE?

BPE was first introduced in Gage [1994] as a method of compressing data by replacing frequently occurring byte pairs with a new byte that was not in the original data. This work inspired NLP researchers, who adopted and modified the original compression algorithm as a method of tokenizing text. The NLP version of BPE follows the same compressive principles of the original algorithm, by merging frequently occuring subwords into new tokens. Consider the toy example text: "*ababca*". BPE begins with a token for each atomic subword unit. In this case, these are the characters '*a*', '*b*', and '*c*'. Then, the algorithm iterates over the text to find the most frequently occuring pair of tokens. In this example, the most frequently occuring pair would be "*ab*", which is added to our vocabulary as a new token. This process is greedily repeated until we reach a given number of tokens in our vocabulary.

Prior to the widespread usage of BPE for tokenization in NLP, the two predominant tokenization strategies were world-level tokenization and character-level tokenization. Word-level tokenization learns embeddings for each word in the tokenizer's vocabulary. This strategy was very effective at encapsulating the meaning of a word by tokenizing each word as a separate token. However, if the text included words outside of the model's original vocabulary, then the word would receive a designated "unknown" token. This poor handling of unknown values bottlenecks information and results in poor generalization to new distributions of text. By contrast, character-level tokenization learns embeddings for each individual character in the training text. In doing so, character-level tokenization is capable of handling any text, as the text is segmented into each individual character. However, this approach also bottlenecks information, as it assigns an identical embedding to each instance of the same character (e.g. '*e*' has the same value in "*weapon*", "*peace*", and "*pterodactyl*"). This approach does not consider how characters may have drastically different significance depending on the context of the words they are a part of. BPE seemingly solved this trade-off by employing the flexibility of character-level tokenization (as it begins with a token for each character), and the contextual information of word-level tokenization (as it iteratively builds subword tokens). This balance made BPE an attractive choice for popular language models, such as BERT (Devlin et al. [2019]) and GPT (Brown et al. [2020]).

However, recent empirical results have indicated that BPE may not be as effective as other tokenization strategies. For example, Gow-Smith et al. [2022] was able to improve performance on downstream tasks by always tokenizing each whitespace as a separate token, in comparison to normal BPE or Unigram tokenizers, which may merge whitespace with surrounding characters. Another issue arises for low-resource languages, which are less represented during the BPE training phase, and result in fewer subword embeddings and poorer tokenization (Zhang et al. [2022]). Finally, Park et al. [2020] analyzes the effect of morphologically segmenting words, followed by applying BPE to each individual morpheme. Using their new tokenization strategy, the authors observed increased performance on English-Korean translation tasks. These results seemingly imply that, by using an alternative non-BPE tokenization algorithm, we could improve overall performance. Specifically, inspired by the results of Park et al. [2020] for English-Korean translation, we aim to investigate the effects of linguistically-informed tokenization for machine translation of different languages.

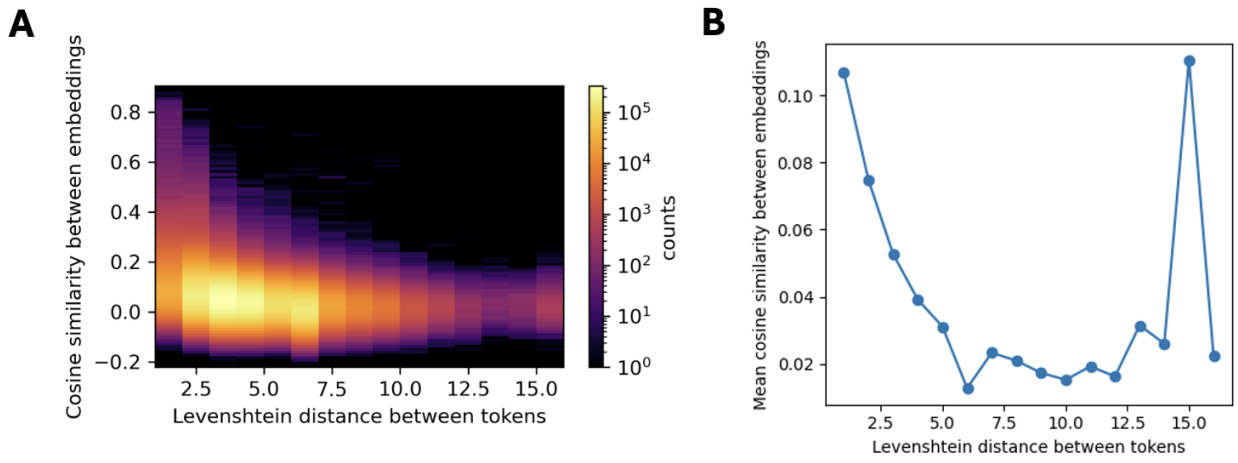

Figure 1: **Cosine similarity between embeddings generally decreases on average with increased edit distance between tokens. A** is binned cosine simiarity vs. edit distance for each unqiue pair of the first 5000 tokens in mT5-small's vocabulary. **B** shows the column-wise average of A, with decreasing average cosine similarity up to edit distance 10, and a max average similarity at distance 15.

2.2 Alternative tokenization

We have established the shortcomings of BPE, however, we aim to construct a more general tokenization strategy to improve performance on the machine translation task. We experiment with two strategies to improve performance for machine translation.

Our first approach is a mild variation of BPE that seeks to address the aforementioned issue of language imbalanace. As previously mentioned, training a BPE tokenizer on an unbalanced multilingual corpus may lead to poorer performance on lower-resource languages. To counteract this effect, we construct a balanced corpus of text from our source and target languages. Then, we train a new "language-specific" BPE tokenizer on this combined corpus. In doing so, we hope to mitigate the effects of language imbalance, while maintaining the relative simplicity of BPE.

Although the above approach resolves the issue of language imbalance, it is still a linguistically-uninformed algorithm. Consequently, it may experience similar shortcomings as normal BPE, as the algorithm attempts to minimize the number of tokens, rather than maximize linguistic information. To design a more linguistically-informed approach, we turn to the field of linguistic morphology. Motivated by previous work with English-Korean translation, we use a morpheme database, known as MorphyNet, (Batsuren et al. [2021]) to decompose a word into its morphemes. As a result, our tokenization is representative of each atomic unit in the text. In doing so, we hope to maximize the linguistic information in our translation.

3 Empirical Section(s)

3.1 Datasets

For both training and evaluation, we use disjoint subsets of the ParaCrawl dataset (Bañón et al. [2020]). ParaCrawl is a parallel dataset featuring various language pairs, primarily aligned with English. The authors used Bitextor (Esplà-Gomis [2009]) – a tool for automatically scraping bitexts from multilingual websites – to download, preprocess, and align documents. Additionally, the authors use Bicleaner (Sánchez-Cartagena et al.) to remove noisy parallel sentence pairs.

For our purposes, we use the English-German (278.3 million sentences), English-Finnish (31.3 million

sentences), and English-Russian (5.4 million sentences) parallel datasets. Due to the varying dataset sizes, we use a random subset of 5 million parallel sentences from each language. In doing so, our evaluation between language pairs is more fair, as we are not potentially conflating our results with a language pair simply having higher resources.

For each language pair, we split the parallel datasets into 4.5 million sentence pairs for training and 500 000 sentence pairs for evaluation. We format the input sentence as "translate English to `{target-language}`: `{english-sentence}`", with the translated sentence being the target output, to align with the original mT5 paper (Xue et al. [2021]).

For our morphologically-informed tokenizers, we use MorphyNet as a source of morphemes for each language in our translation tasks (Batsuren et al. [2021]). MorphyNet is a multilingual database of derivational and inflectional morphology, which we use to help segment words into their morpheme-based tokens. For example, the Finnish word "*tahmauduttaneen*" is segmented as "*tahmaudu | tta | neen*".

For our evaluation metrics, we use both BLEU score (Papineni et al. [2002]) and chrF++ (Popović [2015]). The BLEU score compares the word-level n-grams of machine translations against the n-grams of the target human translations. This metric is very widely used in NLP, but has also been criticized for its poor evaluation on high-morphology languages. High-morphology languages often combine morphemes to create compound words. However, BLEU score fails to capture "partial matches", as it only concerns full words. In recent years, the NLP community has used chrF++ to counterbalance this effect, as chrF++ uses character-level n-grams, which are more granular and may capture more morphological information. For this reason, we include both metrics in our analysis.

3.2 Token edit distance vs. embedding similarity [Kai]

One concern with changing the tokenization of finetuned models is whether small segmentation differences (e.g., "*ally*" as "*al*", "*ly*" vs. "*all*", "*y*") will result in vastly different input embeddings. If the inputs are changed signifigantly from our model's pretraining, we expect performance will sharply decrease. In this section, we analyze this question by comparing the edit distance of tokens in mT5-small's vocabulary with the similarity of their input embeddings.

Method For each unique pair of tokens in the top-5000 from mT5-small's vocabulary, we measure Levenshtein edit distance between strings [Levenshtein et al., 1966] and cosine similarity between their corresponding embeddings.

Results Figure 1 shows binned and mean edit distances vs. cosine similarities for each of the 12.5 million pairs of tokens (excluding pairs of the same token). We find that similarity between embeddings generally decreases as edit distance between tokens increases, with a Pearson correlation of $r = -0.333$ ($p < 10^{-16}$). This indicates that adjusting tokenization to reflect morpheme boundaries may result in similar input embeddings if the tokens are only changed by a few characters. At the same time, we note that the vast majority of token pairs have dissimilar embeddings, with an average cosine similarity of only 0.106 for tokens one edit distance apart. This suggests we must update our models' embeddings during finetuning alongside the tokenization to improve on downstream tasks. Strangely, while mean cosine similarity decreases steadily from edit distances of 1 to 10, we find that the highest mean similarity occurs at an edit distance of 15, possibly due to variance since there are few examples. Next, we test whether this connection between embedding and token similarity translates to downstream translation performance by altering mT5-small's pretrained tokenization.

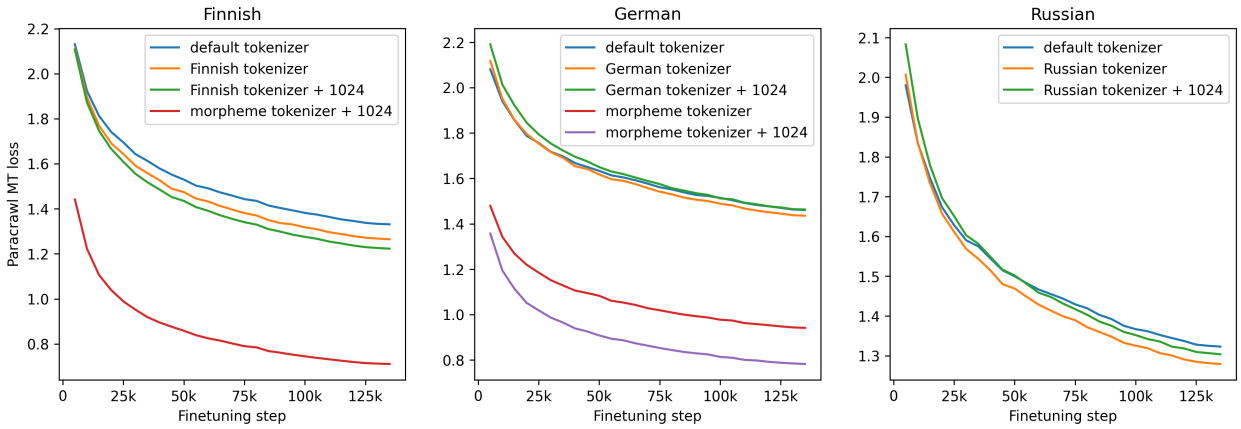

Figure 2: **Translation loss is lower with language-specific tokenizers, and with tokens added to the model's vocabulary.** mT5-small cross-entropy finetuning losses for each langauge and tokenization strategy.

3.3 Finetuning with language-specific and morpheme tokenizers

Method The two modified tokenizers were a language specific tokenizer and a morpheme tokenizer. The language specific tokenizer was simply the mT5's tokenizer retrained on new data comprised of one million English sentences and one million sentences from the target language. Retraining a mT5 tokenizer on two million sentences takes about 30 minutes on CPU.

Creating a morpheme tokenizer is a two step process where we (i) train a tokenizer on all the training data from the target language, and then (ii) split each word in the training data into our algorithm's best guess of where the morpheme boundaries are. Training a tokenizer on 4.5 million sentences takes about 75 minutes on CPU. The morpheme tokenizer is not trained on English since any unknown tokens will be broken up into tokens in the mT5 vocabulary either way.

We created a greedy morpheme tokenizer using the MorphyNet dataset (Batsuren et al. [2021]. The algorithm worked by first creating an affix dictionary that is created by iterating through each row for a target language in the MorphyNet database and adding the affix(es) to a counter that keeps track of the prevalence of each affix. When encountering a new word, the algorithm first searches the MorphyNet database to see if the word already has a predefined morpheme parsing. If it does, the algorithm returns the morpheme parsing. If it doesn't, the algorithm tries to match morphemes from the counter with the target word until it finds a match, at which point it splits the word into the stem and affix. The algorithm does not recursively split the stem due to preliminary experimental results finding that doing so results in oversegmentation. We intentionally chose to create a greedy morpheme tokenizer as opposed to training a model to do it for us since we had MorphyNet at our disposal and our literature review failed to reveal any papers published on morpheme parsers that take a set of morphemes as a parameter.

To overcome the limitations of the HuggingFace Tokenizer module, the algorithm is first used to preprocess the training data. Preprocessing consists of introducing "~" between each best guess morpheme. Preprocessing 5 million sentences takes about 10 hours on CPU. A pretokenizer is used to segment the preprocessed text into the algorithm's best guess morphemes by splitting on "~" and then the pretrained tokenizer tokenizes any morphemes which do not exist the mT5's vocabulary. Preprocessing the training data is required since there is no way to create a Tokenizer object that splits morphemes, so instead we use the rule based morpheme tokenizer to preprocess the training data and set a custom pretokenizer to split any incoming data on "~". Setting a custom pretokenizer requires constructing a Tokenizer object from scratch and is not

Finetuning Tokenizer	Finnish	German	Russian
Pretrained	**20.80** (**41.97**)	**18.77** (**39.84**)	**13.37** (**33.28**)
Language-specific	20.18 (40.24)	**18.68** (**39.96**)	12.97 (31.60)
Language-specific (Added tokens)	17.80 (38.39)	18.20 (39.34)	12.62 (31.29)
Morpheme	DNF	DNF	N/A
Morpheme (Added tokens)	6.91 (25.74)	5.48 (27.39)	N/A

Table 1: **Using language-specific tokenizers and adding tokens to the model's vocabulary prior to finetuning both harm downstream translation performance.** BLEU ↑ scores with chrF++ ↑ in parentheses for each langauge and tokenization strategy. We were unable to train morpheme tokenizers for Russian, and DNF denotes training runs we did not have time to finish.

compatible with simply retraining an existing one, like with the language-specific tokenizers. Because of this, we had to choose from one of the preset tokenizers, and chose GPT2 since it is fundamentally also a BPE tokenizer whose only difference is with how it handles whitespaces, which we are able to remedy in a layer after the tokenizer.

We finetune mT5-small on 4.5 million paracrawl examples for each language using a learning rate of $8e-4$, batch size of 16, and two gradient accumulation steps. We first tokenize samples using either the language-specific or morpheme tokenizers, and then apply the default tokenizer to any tokens which are not in the model's vocabulary. With the Finnish-specific tokenizer, for example, we first tokenize '*Persoonallinen*' as ['*_Persoonalli*', '*nen*'], and then further split the out-of-vocabulary '*_Persoonalli*' into ['*_Perso*', '*on*', '*alli*']. We also experiment with adding the top-1024 unknown tokens from the current language-specific or morpheme tokenizer to the model's vocabulary prior to training. During training, we evaluate loss on a held-out development set of 25000 samples. We used a single l40 or a40 GPU for each of our finetuning runs. Training with default / language-specific and morpheme tokenizers took an average of 12 and 24 hours respectively for each language. In total, we used approximately 240 GPU hours for this project, excluding initial tests.

Results Figure 2 shows cross-entropy loss during finetuning with each of the tokenization strategies. We find that using language-specific tokenizers reduces loss slightly compared to the mT5-small's pretrained tokenizer, and using morpheme tokenizers almost halves the final loss. We also note that adding the top-1024 tokens to the model's vocabulary decreased loss for both language-specific and morpheme tokenization.

Despite lower training loss, each of our new tokenization strategies appear to harm downstream translation performance. With the exception of German, using langauge-specific tokenizers universally lowered BLEU and chrF++ scores, and adding tokens to the model's vocabulary prior to finetuning harmed performance even more. Training with morpheme tokenizers and added tokens resulted in significantly worse scores for German and Finnish, with an over three-fold decrease in BLEU. Unfortunately, due to out-of-memory errors and time constraints we were not able to finish finetuning with the morpheme tokenizer and no added tokens. We were also unable to train a morpheme tokenizer for Russian because GPT2 is unable to handle Cyrillic script.

The negative language-specific results indicate that altering the tokenization after pretraining is likely to hurt downstream performance, regardless of whether the tokenizer is specified for the task languages. As a result, further experiments will need to pretrain models from scratch to evaluate the effect of linguistically informed tokenization. We attribute our morpheme tokenizer's failure to a mismatch between it's source model (GPT2) and our finetuning model (mT5). During training, we noticed that GPT2's tokenizer incorrectly converted many non-latin characters; for instance, replacing "ä" with "Ã¤". We hypothesize there are other

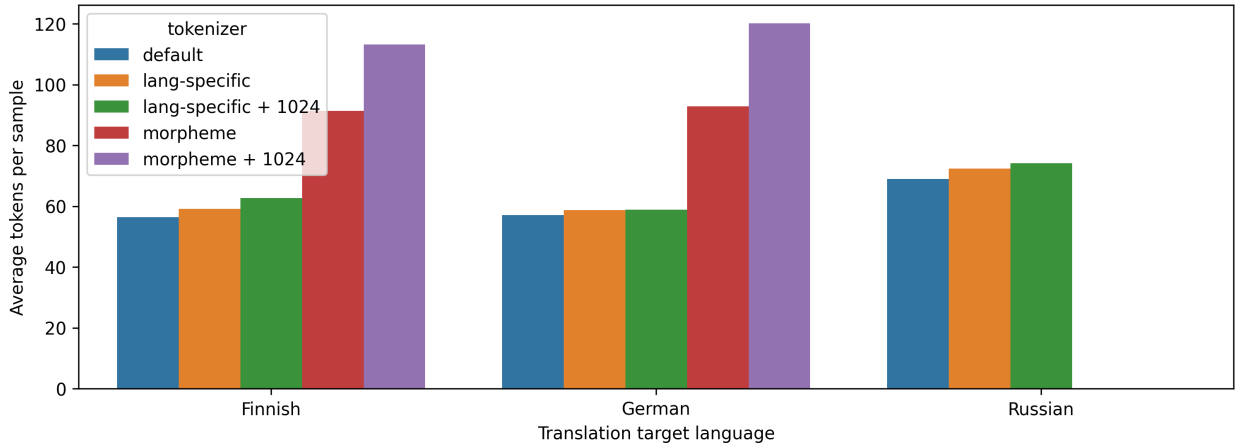

Figure 3: **Language-specific tokenizers and added tokens induce oversegmentation.** Average number of tokens per sample for each langauge and tokenization strategy.

cases of incorrect segmentation which contributed to our model's degradation. This also indicates the disconnect between training loss and downstream performance is caused by tokenization problems rather than better learning of the target language.

We investigate this phenomenon by calculating the average tokens per sample for each language and tokenization strategy, displayed in Figure 3. We find that the language-specific and morpheme tokenizers both induce over-segmentation, mirroring our downstream results. Suprisingly, adding the top-1024 tokens from each tokenizer to the model's vocabulary also increased the average number of tokens in each example, possibly because many additions were short strings like "_-" and "s,.

4 Limitations

Our approach was primarily limited by the computational cost of training and pre-training large models. Ultimately, we are attempting to modify the distribution of tokens with our new tokenization scheme. Consequently, a performance degradation naturally occurs, as we are shifting from the original distribution. However, if we were able to pre-train from scratch (or apply a form of domain-adaptive pre-training), we may have been able to observe improved results, or make a more direct comparison.

Additionally, we used existing databases of morphemes to create the language-specific morpheme tokenizers. As a result, we were limited to the languages that had been added to MorphyNet, and unable to investigate lower-resource (potentially more morphologically complex) languages. Lower-resource languages would likely benefit more from the new tokenization schemes, as they are the most underrepresented in the original BPE tokenization scheme.

5 Future Work

Future research in this direction could directly isolate the issue of tokenization, by pre-training a model from scratch on a large dataset of the source and target languages, using new tokenization strategies. These models could be directly compared against models pre-trained using normal BPE, to investigate whether or not the new tokenization strategies had a positive effect. Our current work does not directly isolate the effects of tokenization, as we use pre-trained models that are aligned with a different (albeit related) distribution.

Similarly, future work could consider the utility of domain-adaptive pre-training (Gururangan et al. [2020]). By using domain-adaptive pre-training, researchers could use larger models and pre-train in a

more cost-efficient manner, rather than needing to restart training from scratch. This task also may involve investigating better methods for handling unknown tokens. Despite our morphological tokenizers, it is still possible to encounter unknown tokens. To shift from the source distribution to the target distribution, we also may require better methods for resolving these tokens.

Finally, future work could try to correlate the performance changes (across a wider group of languages) with linguistic similarity or level of resources (i.e. higher vs lower-resource languages). However, in order to be most effective, it may require further linguistic research to document the morphology of lower-resource languages.

6 Conclusion

In this project we attempt to improve performance on translation tasks by modifying mT5-small's tokenization to reflect the source and target languages. We evaluate using intermediate tokenizers trained on only the source and target languages, and tokenizers trained on to segment words based on their morphemes. Although these alternative tokenization strategies reduce loss when finetuning on translation to the target language, we find that they both induce oversegmentation and harm downstream performance. These results indicate that altering a model's tokenization without changing its embeddings is likely to hurt its translation performance, regardless of whether the tokenizer is specified for the source and target languages. Despite these limitations, improving language-specific tokenization is crucial to reducing gaps in performance and cost between high and low-resource language modelling. As a result, future research should focus on *pretraining* models with linguistically-informed tokenizers.

Acknowledgments

Thank you to Orevaoghene Ahia and Margaret Li for their contributions to this project, especially with regards to their assistance in debugging the morpheme-based tokenizers.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models, 2023.

Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L. Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, Sergio Ortiz Rojas, Leopoldo Pla Sempere, Gema Ramírez-Sánchez, Elsa Sarrías, Marek Strelec, Brian Thompson, William Waites, Dion Wiggins, and Jaume Zaragoza. ParaCrawl: Web-scale acquisition of parallel corpora. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.417. URL https://aclanthology.org/2020.acl-main.417.

Khuyagbaatar Batsuren, Gábor Bella, and Fausto Giunchiglia. MorphyNet: a large multilingual database of derivational and inflectional morphology. In Garrett Nicolai, Kyle Gorman, and Ryan Cotterell, editors, *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 39–48, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.sigmorphon-1.5. URL https://aclanthology.org/2021.sigmorphon-1.5.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Miquel Esplà-Gomis. Bitextor: a free/open-source software to harvest translation memories from multilingual websites. In *Beyond Translation Memories: New Tools for Translators Workshop*, Ottawa, Canada, August 26-30 2009. URL https://aclanthology.org/2009.mtsummit-btm.6.

Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38, 1994. URL https://api.semanticscholar.org/CorpusID:59804030.

Edward Gow-Smith, Harish Tayyar Madabushi, Carolina Scarton, and Aline Villavicencio. Improving tokenisation by alternative treatment of spaces, 2022.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks, 2020.

Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin, editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

Kyubyong Park, Joohong Lee, Seongbo Jang, and Dawoon Jung. An empirical study of tokenization strategies for various korean nlp tasks, 2020.

Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In Ondřej Bojar, Rajan Chatterjee, Christian Federmann, Barry Haddow, Chris Hokamp, Matthias Huck, Varvara Logacheva, and Pavel Pecina, editors, *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL https://aclanthology.org/W15-3049.

Víctor M. Sánchez-Cartagena, Marta Bañón, Sergio Ortiz-Rojas, and Gema Ramírez-Sánchez. Prompsit's submission to wmt 2018 parallel corpus filtering shared task. In *Proceedings of the Third Conference on Machine Translation, Volume 2: Shared Task Papers*, Brussels, Belgium, October . Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer, 2021.

Shiyue Zhang, Vishrav Chaudhary, Naman Goyal, James Cross, Guillaume Wenzek, Mohit Bansal, and Francisco Guzman. How robust is neural machine translation to language imbalance in multilingual tokenizer training? In Kevin Duh and Francisco Guzmán, editors, *Proceedings of the 15th biennial conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 97–116, Orlando, USA, September 2022. Association for Machine Translation in the Americas. URL https://aclanthology.org/2022.amta-research.8.