How do different tokenizers perform on downstream tasks in scriptio continua languages?: A case study in Japanese

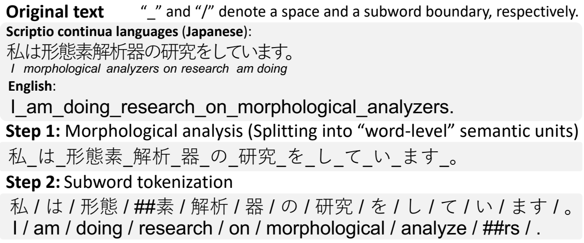
Takuro Fujii[*]
Yokohama National University
tkr.fujii.ynu@gmail.com

Koki Shibata[*]
University of Tsukuba
s1811496@klis.tsukuba.ac.jp

Atsuki Yamaguchi, **Terufumi Morishita** and **Yasuhiro Sogawa**
Hitachi, Ltd.
{atsuki.yamaguchi.xn,terufumi.morishita.wp,yasuhiro.sogawa.tp}@hitachi.com

Abstract

This paper investigates the effect of tokenizers on the downstream performance of pretrained language models (PLMs) in *scriptio continua* languages where no explicit spaces exist between words, using Japanese as a case study. The tokenizer for such languages often consists of a morphological analyzer and a subword tokenizer, requiring us to conduct a comprehensive study of all possible pairs. However, previous studies lack this comprehensiveness. We therefore train extensive sets of tokenizers, build a PLM using each, and measure the downstream performance on a wide range of tasks. Our results demonstrate that each downstream task has a different optimal morphological analyzer, and that it is better to use Byte-Pair-Encoding or Unigram rather than WordPiece as a subword tokenizer, regardless of the type of task.



Original text “_” and “/” denote a space and a subword boundary, respectively.

Scriptio continua languages (Japanese):
私は形態素解析器の研究をしています。
I morphological analyzers on research am doing

English:
I_am_doing_research_on_morphological_analyzers.

Step 1: Morphological analysis (Splitting into “word-level” semantic units)
私_は_形態素_解析_器_の_研究_を_し_て_い_ます_。

Step 2: Subword tokenization
私 / は / 形態 / ##素 / 解析 / 器 / の / 研究 / を / し / て / い / ます / 。
I / am / doing / research / on / morphological / analyze / ##rs / .

Figure 1: Typical tokenization procedures in both *scriptio continua* languages and English

1 Introduction

Tokenization is the first key procedure in current natural language processing when inputting a target sentence to a pretrained language model (PLM). It generally splits an input sequence into subword units, where a subword is a fraction of a word. Previous efforts have proposed several subword-tokenization algorithms (hereafter, subword tokenizers), such as Byte-Pair-Encoding (BPE) (Sennrich et al., 2016), WordPiece (Schuster and Nakajima, 2012), and Unigram (Kudo, 2018), and different PLMs use different subword tokenizers.[1]

It is widely acknowledged that tokenization affects the downstrem performance of PLMs (Rust et al., 2021; Gow-Smith et al., 2022; Bostrom and Durrett, 2020; Park et al., 2020; Toraman et al., 2022). The majority of the previous studies have focused on languages with explicit word boundaries, such as English, while research on *scriptio continua* languages, or languages without word boundaries (like Japanese, Chinese, and Thai), is still understudied. The tokenization process in scriptio continua languages traditionally involves morphological analysis, which splits the input text into morphemes (semantic units similar to words in English) using the dictionary designed by human experts (see Step 1 in Figure 1 for an example). In this case, a tokenizer for a PLM consists of a morphological analyzer and a subword tokenizer. To investigate the impact of tokenization in this scenario, we need to perform a comprehensive study on several sets of the available pairs, which is lacking in the previous work (Bostrom and Durrett, 2020; Inoue et al., 2022; Lowphansirikul et al., 2021).

In this paper, we investigate the effect of tokenizers on the downstream performance of PLMs in scriptio continua languages, focusing on Japanese as a case study. We train an extensive collection of tokenizers consisting of known morphological analyzer and subword tokenizer pairs, use them to pretrain and fine-tune BERT models, and measure their performance on a variety of downstream tasks. On the basis of the experimental results, we address the following three research questions. We first try to answer if we should use a morphological analyzer[2] in a scriptio continua language (Japanese)

[*] Work done while interning at Hitachi, Ltd.

[1] For example, BERT (Devlin et al., 2019) uses WordPiece, and GPT-3 (Brown et al., 2020) uses byte-level BPE.

[2] Not using a morphological analyzer means that we apply subword tokenization directly, the same as in cross-lingual PLMs such as XLM-R (Conneau et al., 2020).

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics - Student Research Workshop, pages 39–49
July 10-12, 2023 ©2023 Association for Computational Linguistics

(RQ1). RQ2 and RQ3 each examine whether different morphological analyzers/subword tokenizers perform differently on a downstream task.

Contributions 1) We test a comprehensive set of known morphological analyzer and subword tokenizer pairs and use various downstream tasks to clarify the effect of tokenizers on the downstream performance of Japanese PLMs. 2) Accordingly, we find the followings:

- We should use a morphological analyzer for Japanese.
- Each task seems to have its own optimal morphological analyzer(s).
- It is better to use either BPE or Unigram as a subword tokenizer rather than WordPiece.

3) We publicly release the code and PLMs.[3]

2 Japanese Tokenizer

In this section, we explain the morphological analyzers and subword tokenizers used in this paper.

2.1 Japanese Morphological Analyzers

Japanese morphological analyzers are based on either a pointwise or sequence prediction method. The former tokenizes a sentence by extracting features from the characters within a pre-defined window and then predicting if a boundary exists between each character using a classifier. The latter first constructs a lattice from an input sentence on the basis of a pre-defined dictionary, where each path in the lattice represents a candidate token sequence and has a cost, and then selects the path with the lowest cumulative cost as the analysis result.[4] We obtain a cost for each path using a statistical model(s) or a hand-crafted dictionary.

We test the following four widely used morphological analyzers: MeCab Ⓜ (Kudo et al., 2004), Juman++ Ⓙ (Tolmachev et al., 2018), Sudachi Ⓢ (Takaoka et al., 2018), and Vaporetto Ⓥ (Akabe et al., 2022). The first three adopt sequence prediction while the last uses pointwise prediction.[5]

2.2 Subword Tokenizers

We compare the following three tokenizers: BPE ($\mathcal{B}$), WordPiece ($\mathcal{W}$), and Unigram ($\mathcal{U}$), each of which differs in either vocabulary construction, tokenization algorithms, or both. These tokenizers are empirically known to produce different subword boundaries (Bostrom and Durrett, 2020).

Vocabulary Construction BPE constructs the vocabulary by merging and adding a pair of existing tokens with the highest score in the dictionary until the total number of tokens in the dictionary reaches a pre-defined size. The score is calculated based on the frequency of the existing tokens. WordPiece is similar to BPE but calculates the score based on the frequency of a symbol pair and the individual frequencies. Unigram heuristically builds a large seed vocabulary from a training corpus (e.g., by taking the most frequent substrings) and then iteratively removes the least important symbols from the vocabulary. Specifically, it first fits a unigram LM for the current vocabulary and then computes (i) the log likelihood of the training corpus with the LM and (ii) that of the training corpus with the LM after removing a particular symbol. It then sets (i) − (ii) as the cost, which shows the degradation of the log likelihood when the symbol is removed. Finally, it removes the symbol with the lowest degradation.

Tokenization BPE splits a word into characters and iteratively merges those with the most frequent pair into larger known symbols in the vocabulary. WordPiece[6] splits a word by the longest subword starting at the beginning of the word in the dictionary and continues splitting until its end. Unigram tokenizes a word by performing Viterbi inference to select the maximum likelihood segmentation based on its vocabulary and unigram LM.

3 Experimental Setup[7]

Tokenizers We compared a total of 12 tokenizers (four morphological analyzers and three subword tokenizers), as introduced in §2. We also considered three additional tokenizers not using morphological analyzers. We trained all tokenizers with the vocabulary size of 30k utilizing 10M sentences randomly extracted from Japanese Wikipedia.

Models We used the base configuration of BERT (total parameters: 125M). For each tokenizer, we pretrained BERT for 500k steps with masked language modeling (Devlin et al., 2019) on the Japanese Wikipedia and CC-100 (Conneau et al.,

[3] Available at `https://github.com/hitachi-nlp/compare-ja-tokenizer`.

[4] Since it is intractable to compute costs for all candidate paths, previous studies have used either the Viterbi algorithm (Viterbi, 1967) or beam search to select a path.

[5] For more details, refer to Appendix A.

[6] We follow the longest-match-first strategy used in BERT.

[7] For implementation details, refer to Appendix C.

Tokenizer		**MARC-ja**	**JSTS**	**JNLI**	**JSQuAD**	**JCQA**	**NER**	**UD**	**Avg.**
Subword	Morphological	Accuracy	Spearman	Accuracy	F1	Acc	F1	LAS	
`bert-base-japanese`		95.5±0.1	85.3±0.3	86.8±0.6	86.4±0.2	76.6±0.8	85.6±0.2	93.3±0.1	87.1
	Ⓜ MeCab	95.4±0.2	84.2±0.1	88.0±0.4	90.1±0.3	74.1±0.7	83.7±0.8	93.6±0.1	87.0
	Ⓙ Juman++	95.5±0.1	84.6±0.4	87.6±0.4	90.1±0.2	73.8±0.3	85.1±0.6	93.6±0.1	87.2
BPE	Ⓢ Sudachi	95.5±0.1	84.2±0.2	88.2±0.3	90.2±0.2	74.2±0.6	83.5±0.6	93.8±0.1	87.1
($\mathcal{B}$)	Ⓥ Vaporetto	95.6±0.1	84.8±0.2	87.5±0.3	89.9±0.2	74.2±1.1	84.1±0.9	93.7±0.1	87.1
	Nothing	95.4±0.2	82.8±0.2	87.2±0.2	88.7±0.3	72.8±0.8	62.9±1.1	93.4±0.1	83.3
	MeCab	95.5±0.1	82.4±0.5	87.5±0.3	89.2±0.3	69.8±0.7	84.0±0.9	93.6±0.1	86.0
	Juman++	95.3±0.3	83.3±0.3	87.7±0.2	89.8±0.3	71.1±0.6	84.7±0.5	93.6±0.1	86.5
WordPiece	Sudachi	95.3±0.2	83.7±0.3	87.2±0.4	89.6±0.1	70.0±0.9	82.4±0.6	94.0±0.1	86.0
($\mathcal{W}$)	Vaporetto	95.3±0.2	83.6±0.1	88.0±0.4	89.7±0.2	71.0±0.4	84.0±0.8	93.8±0.1	86.5
	Nothing	85.5±0.0	N/A	55.3±0.0	10.1±0.1	20.0±0.8	0.0±0.0	63.8±0.9	33.5
	MeCab	95.4±0.3	84.6±0.4	88.3±0.4	89.5±0.3	74.5±0.8	83.1±1.0	93.4±0.2	87.0
	Juman++	95.4±0.2	84.3±0.3	87.8±0.3	89.9±0.2	74.9±1.2	84.1±0.4	93.4±0.1	87.1
Unigram	Sudachi	95.6±0.2	84.8±0.5	88.4±0.3	89.9±0.1	74.5±0.6	83.0±1.3	93.7±0.1	87.1
($\mathcal{U}$)	Vaporetto	95.5±0.3	84.6±0.2	87.9±0.3	89.9±0.1	74.3±0.8	84.1±0.4	93.7±0.1	87.1
	Nothing	95.4±0.4	83.9±0.3	87.7±0.8	89.3±0.1	74.6±0.4	76.9±1.0	93.2±0.2	85.9
Statistical test results: Kruskal-Wallis test (Kruskal and Wallis, 1952). ✓ if $p < .05$ otherwise ✗.									
RQ2: ($\mathcal{B}$, $\mathcal{W}$, $\mathcal{U}$)		(✗, ✗, ✗)	(✓, ✓, ✗)	(✓, ✗, ✗)	(✗, ✗, ✗)	(✗, ✓, ✗)	(✓, ✓, ✗)	(✓, ✓, ✓)	
RQ3: (Ⓜ, Ⓙ, Ⓢ, Ⓥ)		(✗, ✗, ✗, ✗)	(✓, ✓, ✓, ✓)	(✗, ✗, ✓, ✗)	(✓, ✗, ✓, ✗)	(✓, ✓, ✓, ✓)	(✗, ✗, ✗, ✗)	(✗, ✗, ✓, ✗)	

Table 1: Results from seven tasks with standard deviations over five runs. JCQA stands for JCommonsenseQA. Values with a wavy line denote the worst results among morphological analyzers with the same subword tokenizer. ✓ indicates that there is statistical significance among (RQ2) morphological analyzers with the same subword tokenizer or (RQ3) subword tokenizers with the same morphological analyzer, while ✗ denotes that there is no statistical significance. For example, (✓, ✗, ✗) in RQ2 indicates that there is statistical significance between different morphological analyzers with BPE, while no statistical significance is observed for WordPiece or Unigram.

2020) datasets, consisting of 2.2 and 1.1M samples each with the maximum length set to 512.

Benchmarks We used the following benchmarks: JGLUE (Kurihara et al., 2022), NER[8], and Universal Dependencies (UD) Japanese-GSD (Asahara et al., 2018).[9] Since the test set for JGLUE is not publicly available, we fine-tuned all models on the training set using five-fold cross-validation and evaluated their performance on the development set. Since the development and test sets are not available for NER, we split the training set into 9:1. We fine-tuned the models with five-fold cross-validation by the former and measured the performance using the latter.

4 Results and Analysis

This section addresses the three RQs raised in §1.

RQ1: Should we use a morphological analyzer? Table 1 lists the results on the seven downstream tasks grouped by subword tokenizer. The average scores across tasks ("Avg.") show that tokenizers without a morphological analyzer ("Nothing") exhibited the worst results among tokenizers with the same subword tokenizer. This trend also generally holds for task-specific results. These results make intuitive sense because a morphological analyzer can provide explicit semantic boundaries of an input text, making the input units for subword tokenization similar to English words (Figure 1). This should help a model to capture the semantic and syntactic information more easily and consequently outperform those that do not use a morphological analyzer. We therefore conclude that we should use a morphological analyzer for Japanese.

In addition to the above, we observe that WordPiece + Nothing produced by far the worst results in all tasks due to the poor tokenization. WordPiece processes a sequence word by word and treats a sequence without a blank as a single word. If it fails to tokenize a particular word, it tokenizes the "whole" as a single `[UNK]` token. Without a morphological analyzer, the length of a word becomes abnormally long, making WordPiece more likely to produce an `[UNK]` token. This means that the majority of an input text will be converted into `[UNK]` tokens, thus losing almost all of the content in the text. In fact, the average sequence length

[8] Dataset: `stockmarkteam/ner-wikipedia-dataset`

[9] We provide the description of each task in Appendix B. For reference, we also measured the performance of `bert-base-japanese`, which uses MeCab and WordPiece.

	JSTS	JNLI	JCQA	NER	UD
BPE	(Ⓥ > Ⓜ) (Ⓥ > Ⓢ)	–	–	(Ⓙ > Ⓢ)	(Ⓢ > Ⓜ) (Ⓢ > Ⓙ)
WordPiece	(Ⓢ > Ⓜ) (Ⓥ > Ⓜ)	–	–	(Ⓙ > Ⓢ)	(Ⓢ > Ⓜ) (Ⓢ > Ⓙ) (Ⓥ > Ⓜ) (Ⓥ > Ⓙ)
Unigram	–	–	–	–	–

Table 2: Combinations of morphological analyzers with statistical significance ($p < .05$, Steel-Dwass test). "–" indicates no statistical significance observed. "Ⓐ > Ⓑ" indicates that morphological analyzer Ⓐ is significantly better than morphological analyzer Ⓑ.

and ratio of [UNK] per sample in pretraining were 1.15 ± 3.28 and $99.8 \pm 4.9\%$, respectively. These caused unstable pretraining (see Appendix D).

Compared with other tasks, Nothing in NER showed a considerable performance degradation with a maximum difference of 22.2 (Juman++ vs. Nothing in BPE). In NER, annotations are word-level and tend to align well with morphemes. Since tokenizers with morphological analyzers split a morpheme into subword tokens, they can produce more linguistically motivated subword segmentation than Nothing, thus giving them an advantage.

RQ2: Do different morphological analyzers perform differently on downstream tasks? Looking at the statistical test results for RQ2 in Table 1[10], we can see that there were significant performance differences between different morphological analyzers with the same subword tokenizers in some tasks, e.g., JSTS, NER, and UD. In other words, different morphological analyzers could perform differently on different downstream tasks.

For tasks with statistical significance, we further ran the Steel-Dwass test (Douglas and Michael, 1991) to see which morphological analyzer had a significant performance difference from the others (Table 2). We can observe task-specific trends for an effective morphological analyzer(s). Specifically, for JSTS, Vaporetto performed well. For NER, Juman++ was effective. For UD, Sudachi performed well. Therefore, each task seems to have its own optimal morphological analyzer(s).

RQ3: Do different subword tokenizers perform differently on downstream tasks? From the statistical test results for RQ3 in Table 1, we observe significant performance differences between subword tokenizers with the same morphological analyzers in some tasks, such as JSTS and JCQA. "Avg." in Table 1 indicates that WordPiece performed poorly, while BPE and Unigram achieved similar results. The results of the Steel-Dwass test (Table 3) also confirmed that WordPiece showed significant performance degradation compared with either BPE, Unigram, or both in some tasks. We did not observe a significant difference between BPE and Unigram across all tasks. Therefore, different subword tokenizers could perform on downstream tasks differently, and it is better to use either BPE or Unigram.

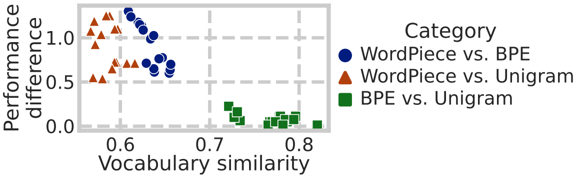


Figure 2: Relationship between vocabulary similarity of subword tokenizers and their performance difference. Samples with the same subword tokenizer are excluded.

We next analyze and discuss which differences in subword tokenizers produced downstream performance differences. First, we look at the difference in the vocabulary of subword tokenizers. We plot the relationship between vocabulary similarity and performance difference between two different subword tokenizers in Figure 2. The vocabulary similarity of two different subword tokenizers is computed as $\frac{|V_1 \cap V_2|}{|V|}$, where $|V|$ is the vocabulary size and V_1 and V_2 are the vocabularies of two subword tokenizers (T_1 and T_2). For each task, we computed the performance difference between the two as $\frac{1}{5}|\sum_i s_{1i} - \sum_j s_{2j}|$, where s_{1i} and s_{2j} are the i-th and j-th observed scores of T_1 and T_2, respectively. We observe that symbols related to WordPiece (● and ▲) are plotted in the upper-left corner, while others (■) are in the lower-right corner, indicating that WordPiece has a different vocabulary composition than BPE and Unigram, and its performance difference is far larger than that between BPE and Unigram. These results are consistent with our finding that WordPiece performed poorly with statistical significance, and both BPE and Unigram showed similar results. Therefore, it is possible that the vocabulary of a subword tokenizer has something to do with the downstream performance.

Further, while WordPiece uses a greedy longest-match-first strategy in tokenizing a word, both BPE

[10] Note that we omit Nothing from the following analyses.

	MARC-ja	JSTS	JNLI	JSQuAD	JCQA	NER	UD
MeCab	–	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	$(\mathcal{B} > \mathcal{W})$	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	–
Juman++	–	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	–	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	–
Sudachi	–	$(\mathcal{U} > \mathcal{W})$	$(\mathcal{U} > \mathcal{W})$	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	$(\mathcal{U} > \mathcal{W})$
Vaporetto	–	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	–	$(\mathcal{B} > \mathcal{W})$ $(\mathcal{U} > \mathcal{W})$	–	–

Table 3: Combinations of subword tokenizers with statistical significance ($p < .05$, Steel-Dwass test). "–" indicates no statistical significance observed. "$\mathcal{X} > \mathcal{Y}$" indicates that subword tokenizer $\mathcal{X}$ is significantly better than subword tokenizer $\mathcal{Y}$.

and Unigram use a more sophisticated approach (as explained in §2.2). This algorithmic difference might also contribute to the performance difference between different subword tokenizers.

5 Conclusion

To investigate the effect of tokenizers on the downstream performance of PLMs in a scriptio continua language (Japanese), we compared extensive sets of tokenizers by evaluating them on a wide range of downstream tasks and addressed the three RQs in §1. Future work will examine how to automatically select the optimal tokenizer pair for a given task.

Limitations

This study has the following limitations:

- We fixed the vocabulary size of each subword tokenizer to 30k. Using a different size might yield different results than those in our paper, though the effect of varying the vocabulary size for a subword tokenizer seemed to be small if the size is sufficiently large (e.g., over 16k or more) (Toraman et al., 2022).
- We have used the BERT architecture for our comparison, while there are other commonly used model architectures such as T5 (Raffel et al., 2020) and GPT-3. The investigation with these architectures is our future work.
- To investigate the impact of tokenizers on the downstream performance of PLMs in scriptio continua languages, we have taken Japanese as a case study. Other scriptio continua languages will be addressed in the future.

Ethics Statement

This study did not involve any sensitive data but only used publicly available data, including Wikipedia, CC-100, JGLUE, Japanese NER, and UD as explained in the paper. Although we plan to release the resulting models, they might perform unfairly in some circumstances, as reported in Baldini et al. (2022). We highly recommend users to refer to studies on debiasing PLMs, such as Guo et al. (2022).

Acknowledgements

We would like to thank anonymous reviewers, Yuta Koreeda, and Yuichi Sasazawa for their insightful comments. We also would like to thank Dr. Masaaki Shimizu for the maintenance and management of the large computational resources used in this paper.

References

Koichi Akabe, Shunsuke Kanda, Yusuke Oda, and Shinsuke Mori. 2022. Vaporetto: Fast japanese tokenizer based on pointwise prediction (in Japanese). In *Proceedings of the 28th Annual Meeting of the Association for Natural Language Processing*.

Masayuki Asahara, Hiroshi Kanayama, Takaaki Tanaka, Yusuke Miyao, Sumire Uematsu, Shinsuke Mori, Yuji Matsumoto, Mai Omura, and Yugo Murawaki. 2018. Universal Dependencies version 2 for Japanese. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Ioana Baldini, Dennis Wei, Karthikeyan Natesan Ramamurthy, Moninder Singh, and Mikhail Yurochkin. 2022. Your fairness may vary: Pretrained language model fairness in toxic text classification. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2245–2262, Dublin, Ireland. Association for Computational Linguistics.

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4617–4624, Online. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

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. 2020. Language models are few-shot learners.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. 2015. Microsoft coco captions: Data collection and evaluation server.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Critchlow E. Douglas and Fligner A. Michael. 1991. On distribution-free multiple comparisons in the one-way analysis of variance. *Communications in Statistics - Theory and Methods*, 20(1):127–139.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

Edward Gow-Smith, Harish Tayyar Madabushi, Carolina Scarton, and Aline Villavicencio. 2022. Improving tokenisation by alternative treatment of spaces. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11430–11443, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yue Guo, Yi Yang, and Ahmed Abbasi. 2022. Autodebias: Debiasing masked language models with automated biased prompts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1012–1023, Dublin, Ireland. Association for Computational Linguistics.

Seiichi Inoue, Nguyen Tung, Akifumi Nakamachi, Shengzhe Li, and Toshinori Sato. 2022. Investigation of the impact of tokenizers using japanese gpt (in Japanese). In *Proceedings of the 28th Annual Meeting of the Association for Natural Language Processing*.

Phillip Keung, Yichao Lu, György Szarvas, and Noah A. Smith. 2020. The multilingual Amazon reviews corpus. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4563–4568, Online. Association for Computational Linguistics.

William H. Kruskal and W. Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 230–237, Barcelona, Spain. Association for Computational Linguistics.

Kentaro Kurihara, Daisuke Kawahara, and Tomohide Shibata. 2022. JGLUE: Japanese general language understanding evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2957–2966, Marseille, France. European Language Resources Association.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Lalita Lowphansirikul, Charin Polpanumas, Nawat Jantrakulchai, and Sarana Nutanong. 2021. Wangchanberta: Pretraining transformer-based thai language models. *CoRR*, abs/2101.09635.

Takashi Miyazaki and Nobuyuki Shimizu. 2016. Cross-lingual image caption generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1780–1790, Berlin, Germany. Association for Computational Linguistics.

Kyubyong Park, Joohong Lee, Seongbo Jang, and Dawoon Jung. 2020. An empirical study of tokenization strategies for various Korean NLP tasks. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 133–142, Suzhou, China. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How good is your tokenizer? on the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Kazuma Takaoka, Sorami Hisamoto, Noriko Kawahara, Miho Sakamoto, Yoshitaka Uchida, and Yuji Matsumoto. 2018. Sudachi: a Japanese tokenizer for business. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Arseny Tolmachev, Daisuke Kawahara, and Sadao Kurohashi. 2018. Juman++: A morphological analysis toolkit for scriptio continua. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 54–59, Brussels, Belgium. Association for Computational Linguistics.

Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. 2022. Impact of tokenization on language models: An analysis for turkish.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira

Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.

Appendices

A Japanese Morphological Analyzers

MeCab (Kudo et al., 2004) MeCab tokenizes a sentence by first constructing a lattice on the basis of its dictionary and then selecting the combination with the lowest cumulative cost using the Viterbi algorithm (Viterbi, 1967). The cost is calculated using a pre-defined feature function in sequence labeling.

Juman++ (Tolmachev et al., 2018) Juman++ tokenizes a sentence by constructing a lattice in accordance with the dictionary and subsequently selecting the path with the highest score by beam search. The score is calculated using both a RNN-based language model and a feature-based linear model.

Sudachi (Takaoka et al., 2018) Sudachi puts an emphasis on offering a tokenizer and dictionary for business use, enabling us to select tokens of different granularity for each application. We use the "Middle" unit of granularity, which is similar to words in general sense.

Vaporetto (Akabe et al., 2022) Vaporetto tokenizes a sentence by extracting features from the characters within a pre-defined window and subsequently classifying if a boundary exists between each character with a linear classification model.

B Downstream Tasks

We briefly describe the seven downstream tasks used in this paper. The statistics for each task dataset are presented in Table 4.

MARC-ja A binary classification task to predict whether a product review is positive or negative. The dataset is based on the Japanese part of the Multilingual Amazon Reviews Corpus (MARC) (Keung et al., 2020).

JSTS A regression task to predict a semantic similarity score between two sentences. The score ranges from 0 (least similar) to 5 (most similar). The data were sourced from the Japanese version of the MS COCO Caption Dataset (Chen et al., 2015) and the YJ Captions Dataset (Miyazaki and Shimizu, 2016).

JNLI A three-way classification task to predict an inference relation between two sentences. The relation includes "contradiction," "neutral," and "entailment," the same as in SNLI (Bowman et al., 2015). The data source was the same as that for JSTS.

JSQuAD A question answering task to predict a corresponding answer span given a question and context. The data were sourced from Japanese articles in Wikipedia and its construction process is based on SQuAD v1.1 (Rajpurkar et al., 2016).

JCommonsenseQA A multiple-choice question answering task to select the best choice from five choices given a question. JCommonsenseQA is a Japanese version of CommonsenseQA (Talmor et al., 2019), and it was constructed in the same manner as in CommonsenseQA, which used the multilingual knowledge base: ConceptNet (Speer et al., 2017) as seeds.

NER A task to identify and categorize named entities in a given sentence. The data were sourced from Japanese articles in Wikipedia and annotated by Stockmark Inc. The dataset is available at `https://github.com/stockmarkteam/ner-wikipedia-dataset`.

UD A dependency parsing task to predict the syntactic dependency structure of a given sentence (Zeman et al., 2017, 2018). The output is a directed tree originating out of a root node. Each edge in the tree has a label that defines a grammatical relationship between two words.

C Implementation Details

We implemented our tokenizers with the Tokenizers library[11] and our models using the PyTorch (Paszke et al., 2019) and Transformers (Wolf et al., 2020) libraries. We trained our models with four NVIDIA V100 (32GB) GPUs for pretraining and one for fine-tuning. We used automatic mixed precision (FP16) provided by PyTorch as default. The code is available on the GitHub: `https://github.com/hitachi-nlp/compare-ja-tokenizer`, and the models are available on the Hugging Face Hub: `https://huggingface.co/hitachi-nlp`.

C.1 Data

We downloaded Wikipedia data from `https://www.tensorflow.org/datasets/catalog/wikipedia#wikipedia20201201ja`. As its preprocessing step, we excluded sentences with less than 30 characters and those containing "`Category`" or table symbols.

[11] `https://github.com/huggingface/tokenizers`

Dataset		License	Task Type	Number of samples		
				Train	Dev	Test
JGLUE	MARC-ja	CC BY-SA 4.0	Text classification	187,528	5,654	-
	JSTS		Sentence pair classification	12,451	1,457	-
	JNLI		Sentence pair classification	20,073	2,434	-
	JSQuAD		Question answering	62,859	4,442	-
	JCommonsenseQA		Question answering	8,939	1,119	-
Japanese NER		CC-BY-SA 3.0	Named entity recognition	5,343	-	-
UD-Japanese-GSD		CC BY-SA 4.0	Dependency parsing	7,050	507	543

Table 4: Statistics for each dataset used in this paper. Note that the test sets are not currently publicly available for JGLUE. Japanese NER does not have the corresponding development and test sets.

Hyperparameter	Value
Batch size	128
Total training steps	500,000
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.999
Sequence length	512
Learning rate	1e-4
Learning rate schedule	Linear warmup
Warmup steps	10,000
Weight decay	0.01
Attention dropout	0.1
Dropout	0.1

Table 5: Hyperparameters for pretraining

C.2 Model

We used the base configuration of BERT (12 hidden layers and attention heads, $\text{Dim}_{\text{hidden}} = 768$, $\text{Dim}_{\text{intermediate}} = 3072$, Total parameters = 125M).

C.3 Pretraining

We pretrained all models for 500k steps and optimized them with AdamW (Loshchilov and Hutter, 2019). We mostly followed the configurations of Devlin et al. (2019). Table 5 lists the hyperparameter settings used in pretraining.

C.4 Fine-tuning

Hyperparameter	Value
Batch size	32
Epochs	5 for JGLUE tasks & NER 10 for UD
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.999
Sequence length	512 for MARC-ja & UD 348 for JSQuAD 128 for JSTS, JNLI & NER 64 for JCQA
Learning rate	3e-5 for JGLUE tasks & NER 5e-5 for BERT in UD 1e-3 for BAP in UD
Learning rate schedule	Linear warmup
Warmup steps	10% of steps
Weight decay	0.01
Attention dropout	0.1
Dropout	0.1

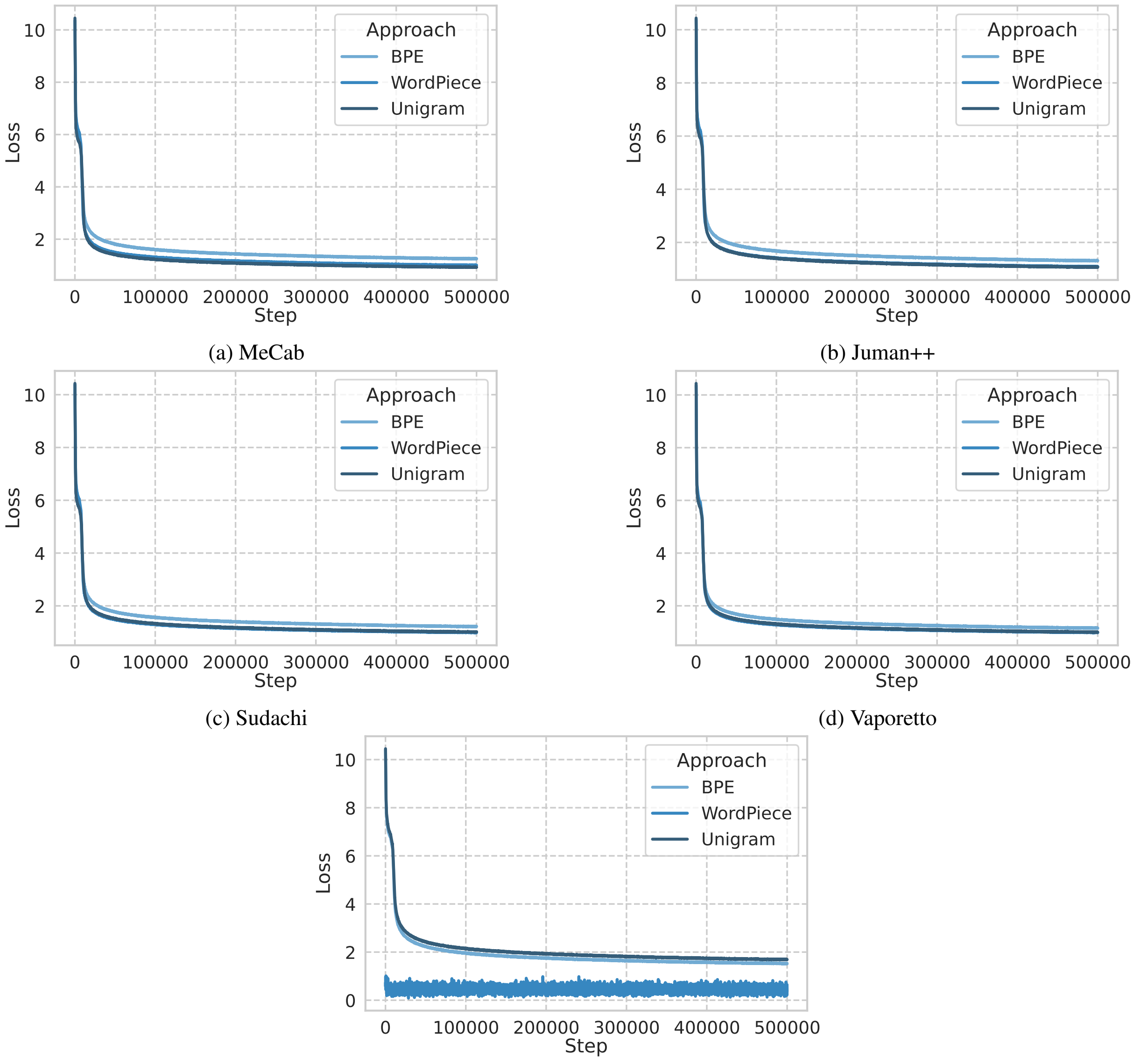
Table 6: Hyperparameters for fine-tuning

Table 6 lists the hyperparameters for fine-tuning models on the JGLUE, NER, and UD datasets. For UD, we trained a deep biaffine attention parser (Dozat and Manning, 2017) built on top of the PLMs. We computed an average for each token over the top four layers of the BERT hidden representations and used it as an input to a biaffine attention parser (BAP). The dimensionalities of arc and relation features given to each biaffine module are 500 and 100, respectively. We used the SuPar library[12] to implement the parser and followed its default hyperparameter configurations.

D Pretraining Loss

Figure 3 shows the pretraining loss curves for our models grouped by morphological analyzer. We can see that WordPiece + Nothing was unstable in pretraining.

[12] https://github.com/yzhangcs/parser



(a) MeCab

(b) Juman++

(c) Sudachi

(d) Vaporetto

(e) Nothing

Figure 3: Pretraining loss curves