Vocabulary Replacement in SentencePiece for Domain Adaptation

Teruno Kajiura
Japan Women's University
Tokyo, Japan
m1816023kt@ug.jwu.ac.jp

Shiho Takano
Japan Women's University
Tokyo, Japan
m1816053ts@ug.jwu.ac.jp

Tatsuya Hiraoka
Tokyo Institute of Technology
Tokyo, Japan
tathi029@gmail.com

Kimio Kuramitsu
Japan Women's University
Tokyo, Japan
kuramitsuk@fc.jwu.ac.jp

Abstract

Large language models (LLMs) show poor performance due to excessive subword segmentation. This problem is particularly evident when the vocabularies of the pretrained domain differ from the domain of the target task. Traditionally, researchers extend the domain-specific vocabulary of the target task by increasing the LLMs' vocabulary size. This approach is problematic because the number of parameters in the LLMs increases significantly. In contrast, we propose a new method that does not change the vocabulary size, instead replacing it with a domain-specific vocabulary. The vocabulary replacement method is proposed as follows. First, we select words that consist of multiple subwords in the vocabulary model. Next, we replace the less frequently used words with ones from the domain-specific vocabulary. Finally, we update the vocabulary model. In this study, we conducted our experiments on the SentencePiece vocabulary model. The vocabulary replacement on the Japanese T5 and multilingual T5 shows that our proposed method produces a space of $6k \sim 21k$ words that can be replaced. In addition, we also compare the performance of the LLMs with Python code generation as the target task. The results showed almost no performance degradation on the Japanese General Language Understanding Evaluation tasks and improved performance of the code generation tasks.

1 Introduction

Recent work has shown that large language models (LLMs) perform well on tasks requiring general knowledge, but have difficulty on domain-specific tasks (Lee et al., 2020; Ebrahimi and Kann, 2021; Tanaka and Shinnou, 2022). The tokenizer segments text based on a vocabulary constructed by extracting words and phrases used in the pre-training corpus. Therefore, if the vocabulary of the pre-trained domain differs from the domain of the target task, the tokenizer often performs excessive subword segmentation of words in the target task (Zhang et al., 2020). One solution is to expand the vocabulary to include target task-specific words in the vocabulary (Zhang et al., 2020; Wang et al., 2020). This approach is superior for improving the targeted downstream tasks. However, it has the problem that if the vocabulary size of the LLMs is increased, the parameters of LLMs increases and the computational efficiency decreases.

In contrast, we propose a new method that does not change the vocabulary size, instead replacing it with a domain-specific vocabulary. The vocabulary replacement method is proposed as follows. First, we select words that consist of multiple subwords in the vocabulary. Next, we replace the less frequently used words with ones from the domain-specific vocabulary. Finally, we update the vocabulary model. In this paper, we replace Japanese vocabulary with Python tokens (reserved words/identifiers) in the Japanese T5 and multilingual T5 (mT5). We confirm the effectiveness of the proposed method by evaluating it on the Japanese General Language Understanding Evaluation (JGLUE) task and the code generation task.

Our main contributions are as follows:

- We proposed a method for domain adaptation of the LLM tokenizer through updates by replacing its vocabulary.
- We have confirmed that high accuracy can be maintained even when replacing 10-20% of the Japanese vocabulary in the LLM.
- Our approach has been empirically proven to enhance the accuracy of code generation

tasks by up to 7%.

Our method shows high applicability across domains and languages for any LLM using SentencePiece.

2 Related Works

This research is closely related to an approach for domain adaptation of pretrained language models and vocabulary expansion.

2.1 Specialized LLM Training

LLMs are often pretrained using large available corpus in general domains (e.g., Wikipedia). However, the use of general LLMs in specialized domains may not be optimal because of differences in language usage and vocabulary.

The two methods in which domain-specific LLMs have been developed are (i) pretraining an LLM from scratch using in-domain data (Lee et al., 2020; Singhal et al., 2023) or (ii) continuing to train existing general LLMs with in-domain data (Gururangan et al., 2020).

Method (i) is a straightforward and effective way to develop domain-specific LLMs, although it has problems that require significant computational resources and lacks a corpus in some domains. Method (ii) modifies these problems and improves the processing performance of using domain-specific knowledge (Yu et al., 2021) and unsupported languages (Chau et al., 2020; Ebrahimi and Kann, 2021) by adapting LLMs.

2.2 Vocabulary Adaptation

The domain characteristics of an LLM are reflected not only in the training sessions (e.g., pretraining and fine-tuning), but also in the LLM vocabulary. The LLM vocabulary functions to split the input text into tokens, which is a format that the LLM can manipulate and understand, and generate embedded representations. The vocabulary of LLMs is determined by tokenization algorithms such as SentencePiece (Kudo and Richardson, 2018) and Byte-Pair-Encoding (Sennrich et al., 2016), as well as by the corpus used for training. Specialized documents contain many unique words.

Vocabulary adaptation to these specialized words is beneficial as a preliminary step when adapting the LLM to a target domain through continued pre-training (method [ii]) (Lamproudis et al., 2022). Zhang et al. expanded the LLM vocabulary to include common words in the domain and achieved improved performance on a Q&A task (Zhang et al., 2020). Additionally, extending an LLM's vocabulary to include words in a new language that it has not previously trained on improves the LLM's ability to process documents in a new language (Wang et al., 2020; Imamura and Sumita, 2022).

3 Proposed Method

In this study, we aim to improve LLM performance on domain-specific tasks through domain adaptation of the vocabulary. We propose a vocabulary replacement method using SentencePiece as a unigram model. SentencePiece (Kudo and Richardson, 2018) is a tokenizer that segments an input string into subwords. It treats high-frequency words as a single piece and splits low-frequency words into shorter subword pieces.

3.1 Idea

We propose a new method for the addition of new words to an LLM's vocabulary for domain adaptation. Figure 1 provides an overview of our proposal. Our method selects some words from the LLM vocabulary and replaces them with new words. After the vocabulary update, we continue pretraining the LLM with a domain-specific corpus in the framework of masked language modeling (Raffel et al., 2020). As shown in Figure 2, the size of the vocabulary increases with each new word added in the vocabulary extension (Zhang et al., 2020; Wang et al., 2020), while our method keeps its size constant.

3.2 Vocabulary Extension

For comparison with our approach, we describe vocabulary extension as done in a previous study. Let the vocabulary of LLM's vocabulary be V_{model}, the vocabulary added through extension be V_{extend}, and the new vocabulary to be added be V_{new}.

The basic vocabulary extension steps are shown as follows.

1. Resize the vocabulary, encoder, and decoder to suit $|V_{new}|$. That is, extend to $|V_{extend}| = |V_{model}|+|V_{new}|$.

2. Continue pretraining with monolingual data of the target language.

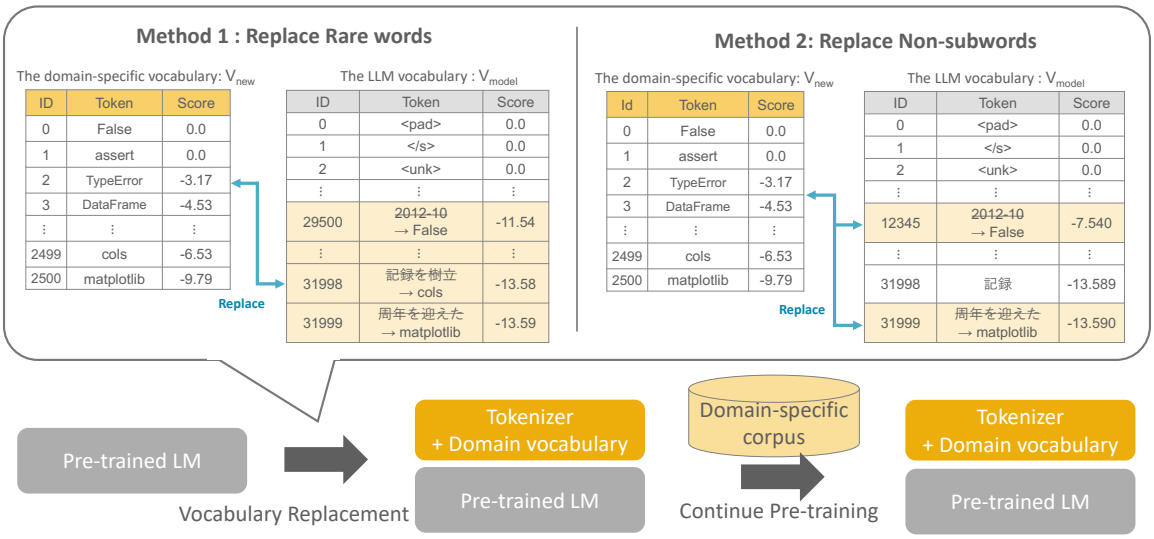

Figure 1: Vocabulary replacement process

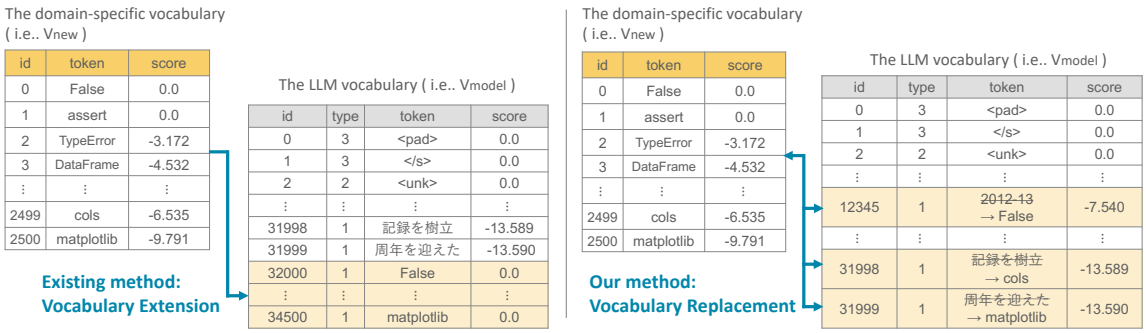

Figure 2: Comparison of vocabulary expansion from prior research (left) and vocabulary replacement proposed in this study (right).

3.3 Vocabulary Replacement

Let the vocabulary to be removed by replacement be V_{remove} and the vocabulary after replacement be V_{remove}. In our method, vocabulary adaptation and training goes as follows:

1. Determine the amount of vocabulary to replace, denoted as $|V_{new}|$, using it as a hyperparameter. Throughout this paper, we followed the $|V_{new}|$ of the previous study (Zhang et al., 2020) and set the sizes to 2.5k and 5k.

2. In the V_{model}, we define the set V_{remove} for replacement, considering the size $|V_{new}|$.

3. Develop a tokenizer by defining $V_{replace}$ that replaces V_{remove} with V_{new}. That is, in the vocabulary after replacement, let V_{remove}=V_{model} + V_{new} − V_{remove}. The size remains the same, $|V_{model}|$=$|V_{remove}|$.

To select V_{remove}, we propose two methods, the Rare Words method and the Non-subword method, which are shown in Figure 1.

Rare Words: We define the low-frequency words in the pre-training corpus as V_{remove}.

Non-subwords: We define V_{remove} as the words that are composed of several subwords and appear infrequently in the pre-training corpus.

3.4 Vocabulary to be Added

In principle, the new vocabulary V_{new} defined for tokenizing can contain any word or subword. However, when choosing V_{new}, it is better to consider how often the word appears in the new domain. We generated V_{new} by using the unigram language model from SentencePiece on the domain-specific corpus for additional training. Our method does not adjust the SentencePiece scores between V_{remove} and V_{new}. We assume that words in both vocabularies are independent, and we replace them based on their scores.

4 Which Words to Replace?

This section describes the non-subword method for selecting replaceable words, separable into subwords (non-subwords) from the model vocabulary V_{model} generated by SentencePiece.

4.1 Sorting of Non-subwords

Let $V = x, y, z$ be a set of tokens that constitute the vocabulary of the model. A non-subword is a word z that has a concatenated relationship with x and y in the form $z = xy$, as in $cook$ and ing. Then, if the relationship between x, y, and z is $x \in V$ and $y \in V$ and $z \in V$, then z is considered a replaceable non-subword.

$$V_{replace} = \{xy | x \in V \cap y \in V\}$$

We consider z to be replaceable, because even if z is excluded from V, z appearing in a sentence can be processed by x and y instead.

SentencePiece adds a score to a word based on its frequency of appearance in the pretrained corpus when generating the vocabulary model. We decide whether to replace the word z in the replaceable subwords, $V_{replace}$, based on the SentencePiece score $p(z)$. In the non-subword method, if the relationship between x and y, which are replaceable subwords, is $p(x) \leqq p(y)$, the word with the lower score $p(x)$ is replaced with V_{new}.

4.2 Replaceable Numerals and Symbols

The vocabulary generated by SentencePiece contains many words with numerals and symbols, as shown in Table 1. We decided to remove repeated numeral and symbol tokens for such words as well as for non-subwords. All numerals are treated as single character tokens; for example, a word such as "2021" is treated as four separate tokens: '2', '0', '2', and '1'. Each symbol is also treated as a single character token. While numeral processing is still debatable, our treatment conforms to the vocabulary structure of the latest LLM, PaLM (Chowdhery et al., 2022).

4.3 Preliminary Experiment

We examined the V_{model} of the mT5 and Japanese T5 [1] to see how many words could be replaced using the non-subword method. In this study, non-subwords are selected from Japanese and numbers/symbols. Table 1 shows the size of $|V_{model}|$, the number of Japanese words in V_{model}, the number of non-subwords in Japanese, and the number of numerals/symbols.

The mT5 supports 101 languages and has a vocabulary size of 250,000. Focusing on Japanese alone, the mT5's V_{model} contains 23,739 words. In contrast, the vocabulary size of the Japanese T5 is 32,128, of which 27,704 are Japanese words. Therefore, the Japanese T5 contains more Japanese vocabulary than the mT5. Using the non-subword method, we found that 21,057 words in the mT5 and 6,228 in Japanese T5 are replaceable. A breakdown shows that the mT5 contains 3,612 Japanese words and 17,445 numbers and symbols. Additionally, the Japanese T5 contains 5,209 Japanese words and 1,019 numbers and symbols.

5 Experiments

We test whether our proposed vocabulary replacement method improves the domain's adaptibility to the task. In this experiment, we evaluate the performance of the Python language as a domain, focusing on the following two points.

- The performance of the model is maintained even when the vocabulary selected by rare word/non-subword is removed.
- Replacement of rare words/non-subwords with Python vocabulary improves the accuracy of the code generation task.

In this experiment, we first replace the Japanese vocabulary with special tokens and evaluate them in JGLUE. JGLUE evaluates the T5's performance in understanding Japanese. Next, we evaluate the effects of replacing the T5 with Python on the accuracy of code generation.

5.1 Setup

Model We use the Japanese T5 and mT5 (Xue et al., 2021) with different vocabulary V_{model} generated by SentencePiece. The Japanese T5 is a T5-adapted language model pre-trained on a Japanese corpus from Wikipedia, OSCAR and CC-100. This model has a vocabulary size of $|V_{model}| =$ 32,128 and consists of 250M parameters. The mT5 is a multilingual T5 pre-trained with the multilingual web text dataset mC4. The size of its vocabulary is $|V_{model}| = 250{,}000$, and the size of the parameters of the small version is 300M.

Python Dataset After vocabulary replacement in Python, we use two unlabeled domain-specific

[1] https://huggingface.co/sonoisa/t5-base-japanese-adapt

Table 1: Analysis of the vocabulary V_{model} in mT5 and Japanese T5.

Model	Vocabulary size (V_{model})	Vocabulary (Japanese)	Non-subword (Japanese)	Numerals & Symbols	Replaceable words
mT5	32,128	27,704	5,209	1,100	6,309
Japanese T5	250,112	23,739	3,612	17,526	21,138

datasets during continual pre-training: CodeSearchNet (Husain et al., 2019) and CoNaLa (Yin et al., 2018). Hyperparameters are 3e-4 peak learning rate and 5 epochs.

Python Vocabulary We add the Python vocabulary as a new vocabulary, V_{new}. Specifically, the Python vocabulary is selected from reserved words and identifiers [2] as well as vocabulary appearing in the Python dataset that scores well on SentencePiece.

5.2 Effect of Removing Words from V_{model}

Our proposed method, by the nature of replacing existing words with new words, means that there is a removed vocabulary, V_{remove}, from the original vocabulary, V_{model}. Removing V_{remove} from the Japanese V_{model} renders the contextual knowledge learned with V_{remove} unusable. This process causes poor performance, especially for Japanese.

We apply two approaches to select V_{remove}: rare word and non-subwords. We aim to verify that even after removing V_{remove}, the T5's performance in processing Japanese remains relatively stable.

We used the JGLUE benchmark[3] which includes the following five datasets. MARC-ja is a task of classifying product reviews into two categories: positive or negative. JSTS is a task of estimating the semantic textual similarity between two sentences, scored from 0 to 5. JNLI is a task of categorizing relationships between pairs of premise and hypothesis sentences into three classes. JSQuAD is a reading comprehension task where one reads a document and a related question, then extracts the answer phrase from the document. JCommonsenseQA is a question-answering task based on Japanese commonsense knowledge, consisting of a question and five word choices for the answer.

5.3 Effect of Replacing V_{remove} with Python

The purpose of this experiment is to test whether replacing to a Python vocabulary improves the accuracy of the code generation task. We aim to compare models for vocabulary-based domain adaptation: a baseline with no changes, one using the EXTEND method from previous research (Wang et al., 2020), and our methods for replacing rare words and non-subwords.

For fine-tuning, we use the code generation dataset (Obara et al., 2022), which consists of bilingual Japanese and Python codes. For evaluation metrics, we use ExactMatch (EM), which indicates the percentage of perfect matches between the correct code and the generated code, and BLEU score (Papineni et al., 2002), which indicates the degree of similarity. Note that the generated code is normalized using the code formatter Black[4], so that differences in the code layout, such as whitespace, do not affect the evaluation. Therefore, differences in code layout, such as whitespace, do not affect the evaluation.

5.4 Results

5.4.1 Evaluation of Removing Vocabulary

Table 2 shows the JGLUE scores for the T5 without vocabulary processing (baseline) and the T5 with two types of our proposed replacement methods: rare words and the non-subwords. In addition, we show the Japanese vocabulary size of each model and the percentage of remaining vocabulary when compared to the baseline.

In terms of vocabulary changes due to the removal of Japanese vocabulary, the non-subword method removed more Japanese vocabulary in both the mT5 and the Japanese T5. We have concluded that the non-subword method is specifically designed to target and select Japanese words.

The JGLUE score changed due to vocabulary reduction from the baseline, and scores for the mT5 and Japanese T5 decreased slightly. We tested the results of MARC-ja, JSTS, JNLI, and JCommonsenseQA using the Wilcoxon signed-rank test, setting the significance level at 0.05. If the p-value is less than 0.05, there is a statistically significant difference between the baseline and the

[2] https://docs.python.org/3.10/reference/lexical_analysis.html

[3] Eval data from the JGLUE dataset is used as test data.

[4] https://pypi.org/project/black/

Table 2: Evaluation of vocabulary removal methods in the JGLUE benchmark. Results that fall below the baseline accuracy and show a significant difference ($p<0.05$) in the prediction are shown in bold.

Model		Total count of Japanese vocabulary	MARC-ja (Acc)	JSTS (Pearson/Spearman)	JNLI (Acc)	JSQuAD (EM/F1)	JCommonsenseQA (Acc)
mT5	Baseline	23,739(100%)	0.938	0.871/0.831	0.882	0.670/0.831	0.546
	2.5k Rare words	23,400 (99%)	0.946	0.868/0.832	0.878	0.681/0.831	0.521
	5k Rare words	22,923 (97%)	0.951	0.875/0.840	0.880	0.678/0.839	**0.530**
	2.5k Non-subwords	23,199 (98%)	0.947	**0.861/0.822**	0.878	0.671/0.827	0.513
	5k Non-subwords	20,051 (84%)	0.946	0.873/0.842	**0.872**	0.659/0.812	**0.503**
Japanese T5	Baseline	27,704(100%)	0.959	0.898/0.870	0.908	0.720/0.866	0.690
	2.5k Rare words	25,679 (93%)	**0.954**	0.891/0.860	0.906	0.724/0.870	0.676
	5k Rare words	23,444 (85%)	0.957	0.895/0.866	0.913	0.725/0.871	**0.671**
	2.5k Non-subwords	25,501 (92%)	0.957	0.906/0.874	0.899	0.725/0.871	0.664
	5k Non-subwords	22,482 (81%)	0.955	0.906/0.876	0.902	0.705/0.848	0.671

Table 3: Evaluation of Code Generation Tasks from Japanese to Python. Results better than the baseline score are shown in bold and the best score is underlined.

Model		EM	BLEU
mT5	Baseline	20.54	57.51
	Extend 2.5k words	**22.15**	**57.73**
	Extend 5k words	**24.92**	**59.19**
	2.5k Rare words	**23.08**	**58.32**
	5k Rare words	**27.39**	**61.28**
	2.5k Non-subwords	**23.08**	**57.58**
	5k Non-subwords	**26.31**	**61.06**
Japanese T5	Baseline	30.69	61.92
	Extend 2.5k words	24.31	**62.37**
	Extend 5k words	18.77	61.76
	2.5k Rare words	**34.46**	**64.20**
	5k Rare words	**34.54**	**65.41**
	2.5k Non-subwords	**32.00**	**63.97**
	5k Non-subwords	**31.69**	**64.44**

reduced model. Because JSQuAD outputs strings, it was excluded from the test.

The bold texts in Table 2 indicate a statistically superior decrease in score compared to the baseline. With the rare words method, there is a significant decrease in mT5's JCommonsenseQA (5,000 words), the Japanese T5's MARC-ja (2,500 words), and JCommonsenseQA (5,000 words). With the non-subword method, performance on the mT5's JSTS (2,500 words), JNLI (5,000 words), and JCommonsenseQA (5,000 words) decreased significantly. However, there were no corresponding tasks on the Japanese T5 that showed showed this decrease. These results reveal that for mT5, a multilingual model, the rare words method is the vocabulary selection approach that minimally impacts affects the original model's performance. Meanwhile, for the Japanese T5, a model specialized for Japanese, the non-subwords method is the most optimal.

5.4.2 Evaluation of the Downstream Task

Table 3 shows the results of the model for the code generation task using the baseline, the extension, and our proposal.

First, results confirm the effect of increasing words of the domain in the vocabulary. The extend and replace methods improved the mT5 score. Our replace methods also improved performance for the Japanese T5. In contrast, Extend decreased EM scores by 6% $\sim$ 11%. The results suggest that increasing domain-specific vocabulary is generally effective in improving task performance.

Next, we compare three vocabulary adaptation methods. For the mT5 and Japanese T5, models using the Replace methods score higher than extend, with rare words having the most effect. BLEU scores also show the same trend. Even without focusing on subwords patterns, as in non-subwords, it is most effective to replace words that are rare in the corpus, as in rare words.

Then, we confirm the difference by the number of words to be replaced. In the mT5, accuracy tends to increase as more Python tokens are added. However, extend's EM score decreased significantly with extension in the Japanese T5, decreasing by 6.3% for 2,500 words and by 11.9% for 5,000 words.

6 Conclusion

In this study, we propose a vocabulary-based domain adaptation that replaces domain-specific words to improve performance in the target domain-specific task. Our method allows the addition of vocabulary from new domains without increasing the size of the LLM's vocabulary. For the addition of domain-specific words by replacement, we remove words from the original LLM

vocabulary. We propose rare words and non-subwords as candidates for removal, aiming to minimize the impact on LLM's performance.

In our experiments, we applied two replacement methods to the Japanese T5 and the mT5. For the selection of words to be removed for replacement, the rare-word method works well for finding less-used words in multilingual models like the mT5. On the other hand, the non-subword method keeps its performance stable when focusing on one language. It works especially well for the Japanese T5. For the effect in the target domain, our method demonstrated improvements over the baseline accuracy in code generation tasks for the mT5 and the Japanese T5. Notably, our method showed significant improvements in code generation tasks compared to vocabulary extension. This approach can be applied to any LLM using SentencePiece, regardless of domain or language.

In the future, we aim to expand this approach to more domains and explore a more generalizable approach for domain adaptation.

Acknowledgements

This research was supported by joint research with NTT Software Innovation Center and JSPS KAKENHI Grant Number 23K11374.

References

Ethan C Chau, Lucy H Lin, and Noah A Smith. 2020. Parsing with multilingual bert, a small corpus, and a small treebank. *arXiv preprint arXiv:2009.14124*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Abteen Ebrahimi and Katharina Kann. 2021. How to adapt your pretrained multilingual model to 1600 languages. *arXiv preprint arXiv:2106.02124*.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Kenji Imamura and Eiichiro Sumita. 2022. Extending the subwording model of multilingual pretrained models for new languages. *arXiv preprint arXiv:2211.15965*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Anastasios Lamproudis, Aron Henriksson, and Hercules Dalianis. 2022. Vocabulary modifications for domain-adaptive pretraining of clinical language models. In *HEALTHINF*, pages 180–188.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Momoka Obara, Yuka Akinobu, Teruno Kajiura, Shiho Takano, and Kimio Kuramitsu. 2022. A preliminary report on novice programming with natural language translation. In *IFIP WCCE 2022: World Conference on Computers in Education*.

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

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

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.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. 2023. Large language models encode clinical knowledge. *Nature*, pages 1–9.

Hirotaka Tanaka and Hiroyuki Shinnou. 2022. Vocabulary expansion of compound words for domain adaptation of bert. *PACLIC-2022*.

Zihan Wang, Stephen Mayhew, Dan Roth, et al. 2020. Extending multilingual bert to low-resource languages. *arXiv preprint arXiv:2004.13640*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 483–498, Online. Association for Computational Linguistics.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM.

Tiezheng Yu, Zihan Liu, and Pascale Fung. 2021. Adaptsum: Towards low-resource domain adaptation for abstractive summarization. *arXiv preprint arXiv:2103.11332*.

Rong Zhang, Revanth Gangi Reddy, Md Arafat Sultan, Vittorio Castelli, Anthony Ferritto, Radu Florian, Efsun Sarioglu Kayi, Salim Roukos, Avirup Sil, and Todd Ward. 2020. Multi-stage pre-training for low-resource domain adaptation. *arXiv preprint arXiv:2010.05904*.