Sub-Character Tokenization for Chinese Pretrained Language Models

Chenglei Si[1,2*], Zhengyan Zhang[1*], Yingfa Chen[1*], Fanchao Qi[1],
Xiaozhi Wang[1], Zhiyuan Liu[1†], Yasheng Wang[3], Qun Liu[3], Maosong Sun[1†]

[1] NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing, China
`{zy-z19,yingfa-c18,qfc17,wangxz20}@mails.tsinghua.edu.cn`
`{liuzy,sms}@tsinghua.edu.cn`
[2] University of Maryland, College Park, MD, USA
`clsi@terpmail.umd.edu`
[3] Huawei Noah's Ark Lab, Hong Kong, China
`{wangyasheng,qun.liu}@huawei.com`

Abstract

Tokenization is fundamental to pretrained language models (PLMs). Existing tokenization methods for Chinese PLMs typically treat each character as an indivisible token. However, they ignore the unique feature of the Chinese writing system where additional linguistic information exists below the character level, i.e., at the sub-character level. To utilize such information, we propose sub-character (SubChar for short) tokenization. Specifically, we first encode the input text by converting each Chinese character into a short sequence based on its glyph or pronunciation, and then construct the vocabulary based on the encoded text with sub-word segmentation. Experimental results show that SubChar tokenizers have two main advantages over existing tokenizers: 1) They can tokenize inputs into much shorter sequences, thus improving the computational efficiency. 2) Pronunciation-based SubChar tokenizers can encode Chinese homophones into the same transliteration sequences and produce the same tokenization output, hence being robust to homophone typos. At the same time, models trained with SubChar tokenizers perform competitively on downstream tasks. We release our code and models at `https://github.com/thunlp/SubCharTokenization` to facilitate future work.

1 Introduction

Large-scale Transformer-based pretrained language models (PLMs) (Devlin et al., 2019; Liu et al., 2019; Lan et al., 2020; Clark et al., 2020; He et al., 2021, inter alia), in which tokenization

---

[*]Equal contribution.
[†]Corresponding authors.

plays a fundamental role, have achieved great success in recent years and attracted wide research interest.

The most popular type of tokenization adopted by PLMs is sub-word tokenization, such as byte pair encoding (BPE) (Sennrich et al., 2016), Word-Piece (Schuster and Nakajima, 2012), and unigram language model segmentation (Kudo, 2018). Recent Chinese PLMs such as CPM (Zhang et al., 2020, 2021b) adopt this kind of sub-word tokenization. Apart from sub-word tokenization, many other Chinese PLMs adopt a simple character tokenizer (CharTokenizer for short) that treats every single Chinese character as a token (Sun et al., 2019; Cui et al., 2019a, 2020, inter alia).

However, we believe that both of these existing tokenizers are sub-optimal for Chinese. This is based on the observation that Chinese has unique linguistic characteristics:

1) Chinese has an opaque orthography with irregular grapheme-phoneme correspondence (Hao and Yang, 2021). This is in contrast to transparent orthographies like Spanish and Finnish where each letter approximately represents one sound. As a result, utilizing pronunciation information in Chinese requires explicit pronunciation encoding.

2) Chinese does not have morphological inflection, unlike morphologically-rich languages like Russian (Coulmas, 1991). This renders sub-word tokenization less useful since the main advantage of sub-word tokenization comes from the fact that it can split common affixes and root words as separate tokens. In fact, Chinese characters are logograms, and their glyphs (the composition of radicals) also contain rich semantic information, which can only be captured at the sub-character level.

469

我们家乡安徽歙县风景秀丽。

Our hometown, Shexian County, Ahhui Province, has beautiful scenery.

CharTokenizer
我 | 们 | 家 | 乡 | 安 | 徽 | 歙 | 县 | 风 | 景 | 秀 | 丽 | 。 *(13)*

Sub-word
我们 | 家 | 乡 | 安 | 徽 | 歙 | 县 | 风 | 景 | 秀 | 丽 | 。 *(12)*

SubChar-Wubi
q#wu# (我们) | pe#xte# (家乡) | pv#tmgt3# (安徽) | wgk | w | 4# (歙) | egc# (县) | mq#jy#te#gmy0# (风景秀丽) | 。 *(9)*

SubChar-Pinyin
wo③3#men②19# (我们) | jia①13#xiang①1# (家乡) | an①13#hui①12# (安徽) | she④16#xian④22# (歙县) | feng①58#jing③13# (风景) | xiu④8#li④50# (秀丽) | 。 *(7)*
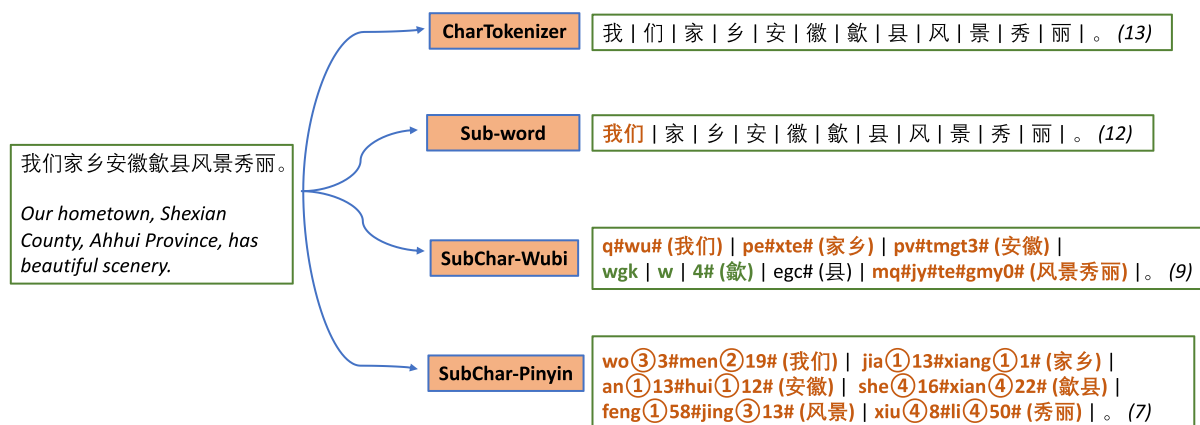
Figure 1: Comparison of existing tokenizers (character tokenizer and sub-word tokenizer) and our sub-character tokenizers (SubChar-Wubi using glyph and SubChar-Pinyin using pronunciation encoding). Different tokens produced by the tokenizers are separated by '|'. The numbers in *(brackets)* indicate the number of tokens in the tokenized sequence. Tokens in orange indicate character combinations, while tokens in green indicate sub-character tokens. '#' indicates the special separation symbol after each character, circled numbers (①②③④) indicate the intonation of characters. (Figure best viewed in color.)

Motivated by these observations, we propose the novel sub-character (SubChar) tokenization (Figure 1). It first encodes every Chinese character into a sequence of phonetic or stroke symbols, and then it uses a sub-word segmenter (such as BPE) to construct the vocabulary on all the encoded sequences. In this way, the resultant tokenizers can capture sub-character tokens that correspond to meaningful phonetic or morphemic units, which are absent from all existing Chinese tokenizers. As far as we know, this is the first attempt on leveraging the sub-character information for language models, especially in the context of Chinese NLP.

To assess the effectiveness of our proposed method, we train a series of BERT-style PLMs using the existing and proposed tokenizers. We evaluate these models on over ten datasets of various downstream natural language understanding (NLU) tasks. Through extensive evaluation, we find that models trained with SubChar tokenizers match models trained with character and sub-word tokenizers on downstream task performance. More importantly, SubChar tokenizers have two major advantages compared to existing tokenizers:

1) **SubChar tokenizers are more efficient.** We find that a small fraction of sub-character tokens in the vocabulary can compose a large variety of rare and complex characters, thus saving much space in the vocabulary for more character combination tokens such as words and phrases.

The increased use of combination tokens leads to significantly decreased length of the tokenized sequences. For example, on the iFLYTEK long text classification dataset, with the same vocabulary size as the CharTokenizers, SubChar tokenizers can achieve as much as 40% length reduction on the tokenized output. Such length reduction can significantly speed up both pretraining and finetuning.

2) **SubChar tokenizers are more robust.** A common and unique type of typos in Chinese is caused by homophones where characters with different semantic meanings have exactly the same pronunciation. SubChar tokenizers based on pronunciation can map homophones into the same transliteration sequences, thus improving robustness against any homophone typos. This could be immensely useful when handling noisy inputs.

We believe that our work is an important step towards more tailored techniques for languages beyond just English by effectively integrating the unique linguistic characteristics of the language (Bender, 2019, #BenderRule).

2 Method

In this section, we describe our proposed SubChar tokenization in detail. We break it down into two steps: 1) Chinese character encoding; 2) vocabulary construction based on the encoded sequences.

470

2.1 Step 1: Character Encoding

The core idea of this step is to encode every Chinese character into a sequence that characterizes its glyph or pronunciation, in order to provide additional inductive biases to the model. We explore several ways of encoding the characters. They can be categorised as pronunciation-based and glyph-based encoding.

Pronunciation-based Encoding In order to capture pronunciation information of characters, we encode Chinese characters using transliteration, which uses IPA-inspired[1] phonetic scripts to characterize the pronunciation.

We explore two different transliteration methods: *pinyin* and *zhuyin* (i.e., *bopomofo*). *Pinyin* uses romanized transcription and four different tones (ˉ, ´, ˇ, `) to transliterate characters, e.g., 魑魅魍魎 → *Chiˉ Mei` Wangˇ Liangˇ*. On the other hand, *zhuyin* uses a set of graphemes non-existent in English and the same four tones to transliterate the characters, e.g., 魑魅魍魎 → ㄔㄇㄟ`ㄨㄤˇㄌㄧㄤˇ. In *zhuyin*, the first tone mark (ˉ) is usually omitted.

We insert special separation symbols (#) after each character's transliterated sequence, e.g., *Chiˉ#Mei`#Wangˇ#Liangˇ#*, ㄔ#ㄇㄟ`#ㄨㄤˇ#ㄌㄧㄤˇ#. This prevents cases where transliterated sequences of different characters are mixed together, especially when there are no tone markers to split them in *zhuyin*.

Different Chinese characters may have the same pronunciation even if they have different semantic meanings (i.e., homophones). For disambiguation, we append different indices after the encoded sequences for the homophonic characters, so as to allow a biunique mapping between each Chinese character and its transliteration sequence, e.g., *Chiˉ33#Mei`24#Wangˇ25#Liangˇ13#*, ㄔ10#ㄇㄟ`3#ㄨㄤˇ6#ㄌㄧㄤˇ1#.

It is unclear whether having such disambiguation of homophones is beneficial or not. To analyze the impact, we also experiment with a variant where we do not add the indices to disambiguate the homophones. We implement the tokenizer *SubChar-Pinyin-NoIndex* to perform *pinyin* encoding without disambiguation indices.

We will show that this variant also has the advantage of being robust to homophone typos (Section 4.2).

Glyph-based Encoding The glyphs (i.e., shapes) of Chinese characters contain rich semantic information and can help NLP models (Cao et al., 2018). Most Chinese characters can be broken down into semantically meaningful radicals. Characters that share common radicals often have related semantic information, e.g., the four characters '魑魅魍魎' share the same radical '鬼' (meaning ''ghost''), and their meanings are indeed all related to ghosts and monsters.[2] In order to capture glyph information, we explore four glyph-based encoding methods, namely, *Stroke, Wubi, Zhengma*, and *Cangjie*.

For *stroke* encoding, we use the Latin alphabet to represent the set of Chinese strokes and convert the characters based on the standard stroke orders,[3] e.g., 魑 → *pszhshpzznnhpnzsszshn*; 魅 → *pszhshpzznhhspn* (underlined parts indicate shared stroke sequences across these characters).

The other three glyph-based encoding methods encode characters into radical sequences instead, by using glyph-based Chinese input methods: *Wubi, Zhengma*, and *Cangjie*. These input methods group strokes together in different ways to form radicals, and then decompose characters into radical sequences. We use the Latin alphabet to represent these radicals, e.g., 魑魅魍魎 → *Wubi: rqcc rqci rqcn rqcw*; *Zhengma: njlz njbk njld njoo*; *Cangjie: hiyub hijd hibtv himob* (underlined parts indicate common radicals among them).

We append the same separation symbol ('#') after each character, and also add the disambiguation indices for characters whose stroke sequences are identical (e.g., 人 (people) and 八 (eight)). However, we note that there are very few cases where different characters have the same glyph encoding.

2.2 Step 2: Vocabulary Construction

Once we have the encoded sequences, we can treat the encoding of each character as the equivalent of 'word' in English and then apply sub-word segmentation to construct the vocabulary for our sub-character tokenizers.

---

[1]IPA: International Phonetic Alphabet (`https://en.wikipedia.org/wiki/International_Phonetic_Alphabet`).

[2]The word '魑魅魍魎' is in fact a Chinese idiom, which is often used to refer to bad people who are like monsters.

[3]`https://en.wikipedia.org/wiki/Stroke_order`.

Sub-word segmentation typically forms sub-word tokens by merging frequent token bigrams, which often results in meaningful morphemes of the words when used in languages like English. On our encoded sequences, sub-word segmentation can capture shared sub-character sequences that correspond to shared radicals or phonetic sequences among similar characters. After running the sub-word segmentation step on the encoded sequences, the vocabulary of the resultant sub-character tokenizers consists of a mixture of sub-character tokens, character tokens, and character combination tokens.

In this work, we use the unigram language model segmentation method (Kudo, 2018) implemented in SentencePiece (Kudo and Richardson, 2018) as the default sub-word segmentation method. In Section 5.6, we also perform an ablation study by setting the sub-word segmentation method to BPE, which results in similar performance and efficiency, illustrating that the gains of SubChar tokenization are insensitive to the specific choice of sub-word segmentation methods.

2.3 Optional Step: Chinese Word Segmentation

Before the first step of character encoding, there is an optional step of Chinese word segmentation.

Chinese word segmentation (CWS) is a common technique to split Chinese text chunks into a sequence of Chinese words. The resultant segmented words sometimes provide better granularity for downstream tasks (Chang et al., 2008). However, the impact of CWS is unclear in the context of pretraining, especially its interplay with the tokenization. Hence, we propose a way to incorporate CWS into our SubChar tokenization and examine whether it is helpful. Our proposed tokenization pipeline is summarized in Figure 2.

Given that the vocabulary of SubChar tokenizers consists of character combinations, characters, and sub-characters, we use CWS to construct the character combination part of the vocabulary. Compared to the character combination tokens generated by the statistical approach of sub-word tokenization, the combination tokens generated by a trained Chinese word segmenter have more linguistic prior knowledge.

Specifically, to construct the vocabulary, we first segment the pretraining corpus into words. Then, we select the most frequent words as the
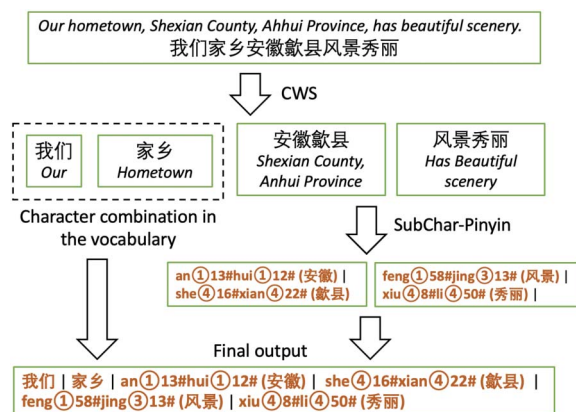

Figure 2: Illustration of the tokenization pipeline when incorporating CWS. After the first step of CWS, high-frequency words (words in the dashed box) directly become part of the final output sequence, the other words then go through SubChar tokenization.

character combination part of the SubChar tokenizer vocabulary. We then encode the text with one of the pronunciation- or glyph-based encoding methods and use sub-word tokenization on the encoded sequences to get the sub-character and character tokens of the vocabulary. Finally, we merge these parts together as the vocabulary for the SubChar tokenizer. When tokenizing new inputs, we first segment them into words, if the words are in the vocabulary, they will be tokenized as word tokens; if not, they will be further processed by the SubChar tokenizer. We control the ratio of word tokens in the vocabulary to be 80% based on preliminary tuning and we use a state-of-the-art segmenter THULAC (Li and Sun, 2009; Sun et al., 2016) for word segmentation.

3 Experiment Setup

In this section, we introduce our baselines, datasets, and experiment settings.

3.1 Baselines

We compare two existing tokenization methods as baselines, namely, single-character tokenization and sub-word tokenization. For a fair comparison, we set the same vocabulary size of 22,675 for all tokenizers, including baselines and our proposed tokenizers. This is consistent with the vocabulary size of Chinese BERT (Devlin et al., 2019).

472

Dataset	#Train	#Dev	#Test
TNEWS	53.4K	10K	10K
IFLYTEK	12.1K	2.6K	2.6K
BQ	100K	10K	10K
THUCNEWS	669K	83.6K	83.6K
CLUEWSC	1.2K	0.3K	0.3K
AFQMC	34.3K	4.3K	3.9K
CSL	20K	3K	3K
OCNLI	45.4K	5K	3K
CHID	519K	57.8K	23K
C3	12K	3.8K	3.9K
CMRC	10K	3.4K	4.9K
CLUENER	11K	1.3K	1.3K

Table 1: Statistics of downstream datasets.

3.2 Pretraining Data

We use the same training corpus to train all the tokenizers in this work. The corpus consists of 2.3 GB Chinese text from Baidu Baike.[4]

To evaluate the effectiveness of the tokenizers, we pretrain a BERT[5] model using each tokenizer and compare their performance on downstream tasks. When pretraining the BERT models, we use the same pretraining corpus (i.e., Baidu Baike) and the same set of hyper-parameters. Notably, we also pretrain a new BERT model using the character tokenizer on our pretraining corpus instead of loading from existing checkpoints (Devlin et al., 2019) so that it provides an apple-to-apple comparison with our proposed methods. Since our proposed tokenizers are direct drop-in replacements for the baseline tokenizers, they do not incur any extra parameters. In summary, all the compared models have the same training corpus, hyper-parameters, and number of parameters, allowing for a truly fair comparison.

3.3 Evaluation Data

We finetune and evaluate the pretrained models with different tokenization methods on various downstream NLU datasets, including single-sentence classification, sentence-pair classification, and reading comprehension tasks. We briefly introduce each dataset below and present the dataset statistics in Table 1.

---

[4]https://baike.baidu.com/.

[5]Note that we mean BERT-style pretrained Transformers. Our models are not directly comparable with the original Chinese BERT since we use different pretraining data and hyper-parameters.

TNEWS (Xu et al., 2020b) is a news title classification dataset containing 15 classes.

IFLYTEK (Xu et al., 2020b) is a long text classification dataset containing 119 classes. The task is to classify mobile applications into corresponding categories given their description.

BQ (Chen et al., 2018) is a sentence-pair question matching dataset extracted from an online bank customer service log. The goal is to evaluate whether two questions are semantically equivalent.

THUCNEWS (Li and Sun, 2007) is a document classification dataset with 14 classes. The task is to classify news into the corresponding categories given their title and content.

CLUEWSC (Xu et al., 2020b) is a coreference resolution dataset in the format of Winograd Schema Challenge (Levesque et al., 2012). The task is to determine whether the given noun and pronoun in the sentence refer to the same entity.

AFQMC (Xu et al., 2020b) is the Ant Financial Question Matching Corpus for the question matching task that aims to predict whether two sentences are semantically equivalent.

CSL[6] is the Chinese Scientific Literature dataset extracted from academic papers. Given an abstract and some keywords, the goal is to determine whether they belong to the same paper. It is formatted as a sentence-pair classification task.

OCNLI (Hu et al., 2020) is a natural language inference dataset. The task is to determine whether the relationship between the hypothesis and premise is entailment, neutral, or contradiction.

CHID (Zheng et al., 2019) is a cloze-style multiple-choice reading comprehension dataset. Given a context where some idioms are masked, the task is to select the appropriate idiom from a list of candidates.

C3 (Sun et al., 2020) is a multiple-choice reading comprehension dataset. The goal is to choose the correct answer for the questions given context.

CMRC (Cui et al., 2019b) is a span-extraction reading comprehension dataset consisting of questions annotated from Wikipedia paragraphs.

---

[6]https://github.com/P01son6415/CSL.

473

CLUENER2020 (Xu et al., 2020a) is a named entity recognition dataset with 10 entity types.

3.4 Hyper-parameters

We elaborate on all hyper-parameters involved for reproducibility (we also release all code, trained tokenizers, and models).

Tokenizer Training. When training tokenizers with SentencePiece, we use a character coverage of 1.0 and model type 'unigram' for all tokenizers being compared. Other hyper-parameters follow the default of SentencePiece.

BERT Pretraining. We follow the training procedure of BERT (Devlin et al., 2019) except that the next sentence prediction objective is removed. The pretraining process consists of two stages. The first stage uses a maximum sequence length of 128 with a batch size of 8K for 8K steps. The second stage uses a maximum sequence length of 512 with a batch size of 4K for 2K steps. We experiment primarily with 6-layer Transformer (Vaswani et al., 2017) models. To ablate the impact of model size, we also pretrain 12-layer Transformer models for the baseline CharTokenizer and proposed SubChar-Pinyin tokenizer. Other model configurations are the same for all models: 12 attention heads, an intermediate size of 3072, and a hidden size of 768.

BERT Finetuning. For the finetuning on downstream datasets, we use a batch size of 32, maximum training epochs of 24, and tune max sequence length in {96, 256, 512}. Since the original test sets are not released, we use the original dev sets as the test sets and randomly hold out 10% of the training set as the dev sets. We select the best checkpoint on the dev sets and report performance on test sets. These hyper-parameters are consistent with previous work. For all experiments in this paper, we report the results of the average run of three different random seeds. All experiments are done on NVIDIA A100 GPUs.

4 Experiment Results

In this section, we present the experiment results and the main findings. We not only evaluate on a wide range of common Chinese NLU datasets, but also perform robustness evaluation on both synthetic and real-world noisy data.

4.1 Standard Evaluation

We compare models trained with our SubChar tokenizers and the baseline tokenizers. There are multiple possible encoding methods for SubChar tokenizers, as described in Section 2. In this section, we choose two representative ones: *Wubi* (glyph-based) and *Pinyin* (pronunciation-based). We later show a full ablation of all different encoding methods in Section 5.5.

Table 2 shows the performance of BERT models with different tokenizers on downstream datasets. Examining the results of the 6-layer BERT models pretrained on the 2.3G Baidu Baike corpus, we observe that despite some variation across different datasets, our proposed sub-character tokenizers can match the baselines on downstream datasets. When scaling the 6-layer models to 12-layer, we observe moderate improvement on the average performance ($70.75 \rightarrow 72.23$ for Char-Tokenizer and $71.42 \rightarrow 72.87$ for SubChar-Pinyin). Besides, we discuss the impact of pretraining data size in Section 5.4. These results demonstrate that on standard NLU benchmarks, our proposed tokenizers can serve as a very strong alternative.

4.2 Robustness Evaluation

Apart from evaluating on the standard benchmarks, we also verify whether our proposed tokenization methods are better at handling noisy inputs. We cover two major Chinese input methods: keyboard input and speech input. For keyboard input, we construct synthetic noise tests via character substitutions. For speech input, we use a noisy test set including inputs with diverse accents, which poses greater typo diversity. Our SubChar-Pinyin method shows advantage in both cases.

Synthetic Typos We simulate the homophone typos that are common in real-world Chinese writing systems, especially user-generated inputs. As shown in Figure 3, *pinyin* input is the most widely used keyboard input method for Chinese users.[7] When users type in the romanization of the intended characters, the input interface will present all Chinese characters with the same romanization for the users to choose from. As a result, it

---

[7]https://en.wikipedia.org/wiki/Chinese_input_methods_for_computers.

	TNEWS	IFLY	THUC	BQ	WSC	AFQMC	CSL	OCNLI	CHID	C3	AVG
					6-layer, 2.3G Corpus						
CharTokenizer	64.19	55.83	96.95	81.99	63.39	68.68	82.67	68.19	72.48	53.17	70.75
	±0.18	±0.50	±0.04	±0.47	±1.95	±0.46	±0.46	±0.39	±0.23	±0.56	±0.31
Sub-word	64.09	54.88	97.14	81.94	62.67	69.25	83.20	69.03	72.78	53.32	70.83
	±0.28	±0.39	±0.03	±0.28	±2.87	±0.42	±0.27	±0.44	±0.13	±0.44	±0.35
SubChar-Wubi	63.89	58.64	97.02	81.70	64.61	68.75	82.81	68.93	72.54	54.68	71.36
	±0.25	±0.27	±0.04	±0.29	±2.09	±0.59	±0.46	±0.38	±0.15	±0.77	±0.23
SubChar-Pinyin	63.68	58.81	97.04	81.74	65.90	68.89	82.87	67.98	73.06	53.03	71.42
	±0.25	±0.28	±0.03	±0.24	±1.45	±0.42	±0.40	±0.45	±0.13	±0.47	±0.19
					12-layer, 2.3G Corpus						
CharTokenizer	64.39	58.52	97.02	83.49	68.09	69.00	82.77	70.40	74.44	54.22	72.23
	±0.13	±0.46	±0.03	±0.38	±1.59	±0.35	±0.33	±0.34	±0.17	±0.40	±0.26
SubChar-Pinyin	64.19	59.67	97.12	82.28	71.71	69.30	82.23	70.43	74.82	55.92	72.87
	±0.14	±0.23	±0.03	±0.16	±2.03	±0.24	±0.27	±0.25	±0.09	±0.26	±0.17
					12-layer, 22.1G Corpus						
CharTokenizer	64.43	59.10	97.12	82.70	70.39	69.39	82.97	69.37	76.34	54.84	72.81
	±0.57	±0.29	±0.01	±0.02	±1.32	±0.06	±0.28	±0.14	±0.62	±1.24	±0.18
SubChar-Pinyin	64.64	59.14	97.10	83.56	72.36	70.67	82.94	69.50	75.92	58.64	73.42
	±0.47	±0.17	±0.04	±0.18	±0.98	±0.66	±0.05	±0.24	±0.45	±0.35	±0.09

Table 2: Results on downstream datasets of different tokenizers. The last column indicates average performance. The subscript is the standard deviation. Models trained with sub-character tokenizers can match the performance of baseline models across all datasets. Ablation shows that increasing the model size or pretraining corpus size can slightly improve downstream task performance. These ablation results support our overall conclusion that models trained with SubChar tokenizers can closely match or slightly outperform the baselines.

Figure 3: An actual interface of the popular *pinyin* input method. The first line *yi yi* is the user input of the romanization sequence, all words with this same pronunciation are listed below for users to choose from.
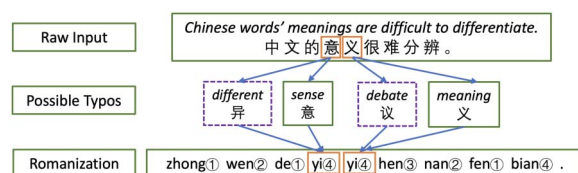

Figure 4: Illustration of how our SubChar-Pinyin-NoIndex tokenizer is robust to any homophone typos. The possible homophone typos (characters in purple dashed boxes) are mapped into the same romanization sequence as the intended correct characters, and hence the resultant tokenization based on the romanized sequences would be the same.

is common for users to choose the wrong characters either by mistake or because they are unclear about the differences among these homophones.

In such cases, our SubChar-Pinyin-NoIndex tokenizer (described in Section 2.1) has the advantage of being robust towards any such homophone typos. As illustrated in Figure 4, the character encoding will map all homophones of a character into the same romanization sequence before undergoing the sub-word tokenization. As a result, the tokenized output will be identical no matter what the typo character is as long as it is a homophone of the intended character.

We inject synthetic noises into the test data and examine whether models trained on clean training data can perform well on these noisy data. To construct the noisy data, we replace the original correct characters with their homophones, e.g., change '意' (sense) to '异' (different)' and '义' (meaning) to '议' (debate).[8] Specifically, we

---

[8]Interestingly, all these four characters have the same pronunciation but different meanings. Moreover, ''意义'' (meaning) and ''异议'' (objection) are homophone words.

TNEWS

	clean	7.5 %	15.0 %	22.5 %	30.0 %	37.5 %
CharTokenizer	**64.10**	63.09	58.96	50.91	38.33	25.20
Sub-word	64.09	62.82	57.75	48.67	36.37	25.72
SubChar-Pinyin	63.68	61.95	56.67	45.22	30.71	27.53
SubChar-Pinyin-NoIndex	63.28	**63.28**	**63.28**	**63.28**	**63.28**	**63.28**

OCNLI

	clean	7.5 %	15.0 %	22.5 %	30.0 %	37.5 %
CharTokenizer	68.37	64.89	56.85	47.65	40.48	36.36
Sub-word	**68.84**	64.33	56.49	48.07	42.68	38.28
SubChar-Pinyin	67.70	61.93	54.39	46.01	40.24	37.33
SubChar-Pinyin-NoIndex	67.91	**67.91**	**67.91**	**67.91**	**67.91**	**67.91**

C3

	clean	7.5 %	15.0 %	22.5 %	30.0 %	37.5 %
CharTokenizer	53.13	51.46	49.22	47.71	46.78	43.95
Sub-word	53.55	51.66	49.49	47.81	46.24	43.58
SubChar-Pinyin	52.87	50.45	47.26	44.50	42.42	40.07
SubChar-Pinyin-NoIndex	**53.65**	**53.65**	**53.65**	**53.65**	**53.65**	**53.65**

Table 3: Results for noisy evaluation with homophone typos. Different columns correspond to different percentages of typos in the test data. The BERT model with our SubChar-Pinyin-NoIndex tokenizer (results in **bold**) suffers no performance drop on noisy test data since it is robust to all homophone typos.

randomly sample a certain ratio $r\%$ of the original characters. For each of them, we replace it with a randomly sampled homophone from all its homophones obtained via a Pinyin dictionary (no replacement if it has no homophones).

The results are shown in Table 3. We observe that there can be a significant drop in performance where there exist homophone typos in the test data. For example, the BERT model trained with CharTokenizer drops from 64.10% accuracy on clean data to 25.20% accuracy when 37.5% of the characters in test inputs are replaced with homophone typos. Overall, we find that the character tokenizer, sub-word tokenizer, as well as the vanilla SubChar-Pinyin tokenizer, cannot handle such noisy data. However, our SubChar-Pinyin-NoIndex tokenizer exhibits **no performance drop** under noises. Moreover, despite learning a shared representation for homophones, the model with SubChar-Pinyin-NoIndex still performs competitively on the clean test sets, either match (on C3) or only a little worse than the baselines (on TNEWS and OCNLI).

Real-World Typos While the above synthetic typos aim to simulate typos in keyboard inputs, another major input method is through speech input where users speak to their devices (like mobile phones) and their speech input is then converted to text for downstream tasks. In order to evaluate model robustness in such scenarios, we use a realistically collected test set that captures such speech input typos. Specifically, we use the speech-noise version of the AFQMC test set from the READIN (Si et al., 2023) benchmark. For each example in this noisy AFQMC test set, three annotators with different accents read the original input, and then the speech recordings are converted to text using commercial automatic speech recognition (ASR) software. We refer readers to the dataset description paper for more data construction details. When computing performance for each test example, we compute both the average across different annotations (Noisy-Average), as well as the worst performance across different annotations (Noisy-Worst), and then take the macro-average across

476

	Clean	N-Avg	N-Worst
CharTokenizer	73.02	44.11	18.81
Sub-word	74.22	42.21	16.91
SubChar-Pinyin	73.24	**45.24**	**19.47**

Table 4: Results on the real-world AFQMC noisy test set. Each clean test instance is annotated by three different annotators, we report both the macro-average on these noisy annotations (N-Average) as well as the average of the worst-case performance across all test examples (N-Worst). SubChar-Pinyin outperforms baselines on the challenging noisy test set (best results on the noisy test set are in **bold**).

all examples. The character-level error rate of the noisy test set is 30% on average.

This AFQMC noisy test set contains not only homophone typos, but also a wide range of other types of real-world input noises due to both the accent variations and ASR errors. The greater diversity of typo types in the real-world test set makes it much more challenging to maintain robustness than the synthetic setting which only considers homophone typos. While the original AFQMC is a binary classification task that classifies whether the question pair is a paraphrase or not, we find that models trained on the AFQMC training set exploit spurious correlations like lexical overlap, even though we explicitly balanced the training set. In particular, when introducing typos in the test data, performance on positive examples drops drastically due to lower lexical overlap, while the performance on negative examples stays or even improves a little because of the lower lexical overlap caused by the typos. This is similar to previous findings on HANS (McCoy et al., 2019) and PAWS (Zhang et al., 2019a). Hence, we follow the evaluation practice when dealing with spurious correlation, which is to focus on improving the worst-group performance, and in this case, we focus on improving performance on the positive examples against the impact of typos.

The results are shown in Table 4, where we report performance on the AFQMC positive examples. All models are trained on the original clean data from AFQMC (we balanced the positive and negative classes during training). We evaluate

on the original clean test set, the Noisy-Average performance (N-Average), and the Noisy-Worst performance (N-Worst). We can see that despite this more challenging speech typo setting, our SubChar-Pinyin model still outperforms the baselines.

These results highlight the robustness advantage of our Sub-Character tokenization method in both dealing with synthetic homophone typos as well as on more diverse real-world typos.

4.3 Effect of CWS

We examine the impact of incorporating CWS in the tokenization as described in Section 2.3. We train tokenizers with and without CWS as preprocessing and compare the performance of the corresponding pretrained models. The results are reported in Table 5. We highlight the take-aways as following: (1) The influence of CWS varies largely across different datasets and tokenizers. Specifically, for the same tokenizer (e.g., SubChar-Wubi), the impact of CWS is positive on some datasets while negative on others; on the same dataset (e.g., TNEWS), the impact of CWS is positive on some tokenizers but negative on others. One of the exceptions is that we observe consistent improvement on AFQMC, which aims to identify whether two sentences are paraphrases. We hypothesize that the fine-grained sentence structures provided by CWS may help the model capture more relevant features. In contrast, we observe consistent degradation on IFLYTEK, which is a long-text classification task. We hypothesize that CWS brings little benefit to long inputs when there is already abundant information. (2) For the overall average performance, only SubChar-Pinyin + CWS achieves slightly better performance than the no-CWS baseline. Out of the seven datasets we evaluated, SubChar-Pinyin + CWS improves the performance on OCNLI (+1.95) and AFQMC (+0.77) which accounts for most of the overall improvement. On the other datasets, CWS either has marginal impact or slightly degrades performance.

Hence, we conclude that adding CWS as an additional step does not consistently help downstream task performance. These results serve as empirical evidence that CWS is not very effective in the use of PLMs, complementary to the results of Li et al. (2019) on models without pretraining.

477

	TNEWS	IFLYTEK	CLUEWSC	AFQMC	CSL	OCNLI	C3	AVG
Sub-word	64.09	54.88	62.67	69.25	83.20	69.03	53.32	65.21
Sub-word + cws	64.26 $_{\uparrow 0.17}$	54.15 $_{\downarrow 0.73}$	63.05 $_{\uparrow 0.38}$	69.62 $_{\uparrow 0.37}$	82.87 $_{\downarrow 0.33}$	68.64 $_{\downarrow 0.39}$	51.77 $_{\downarrow 1.55}$	64.91 $_{\downarrow \mathbf{0.30}}$
SubChar-Wubi	63.89	58.64	64.61	68.75	82.81	68.93	54.68	66.04
SubChar-Wubi + cws	63.57 $_{\downarrow 0.32}$	58.01 $_{\downarrow 0.63}$	64.38 $_{\downarrow 0.23}$	69.41 $_{\uparrow 0.66}$	82.62 $_{\downarrow 0.19}$	69.43 $_{\uparrow 0.50}$	53.15 $_{\downarrow 1.53}$	65.80 $_{\downarrow \mathbf{0.24}}$
SubChar-Pinyin	63.68	58.81	65.90	68.89	82.87	67.98	53.03	65.88
SubChar-Pinyin + cws	63.73 $_{\uparrow 0.05}$	57.89 $_{\downarrow 0.92}$	64.51 $_{\downarrow 1.39}$	69.66 $_{\uparrow 0.77}$	82.90 $_{\uparrow 0.03}$	69.93 $_{\uparrow 1.95}$	53.63 $_{\uparrow 0.60}$	66.04 $_{\uparrow \mathbf{0.16}}$

Table 5: Downstream task results of models trained with different tokenizers. Numbers in subscripts indicate the difference between adding and not adding the cws step in tokenization. Adding cws **does not** bring significant improvement on average.

	CMRC	CLUENER
CharTokenizer	56.58	69.61
Sub-word	55.85	67.94
SubChar-Wubi	54.45	70.63
SubChar-Pinyin	55.18	70.77

Table 6: Results on two character-level classification datasets: CMRC (span-extraction) and CLUENER (named entity recognition). Models are 6-layer BERT. Models with SubChar tokenizers perform close to the baseline models.
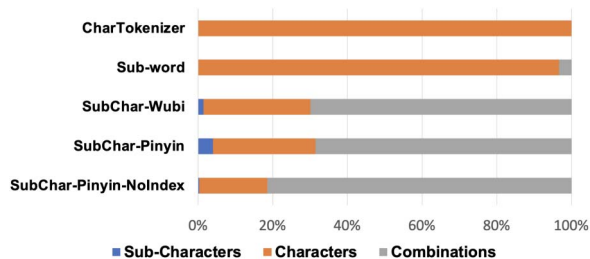

Figure 5: Breakdown of different types of tokens in the vocabularies of various tokenizers. We observe the clear trend that in our SubChar tokenizers, a small fraction of sub-character tokens saves the space to store much more character combination tokens (e.g., words and phrases).

4.4 Character-Level Tasks

The evaluation in Section 4.1 is restricted to sequence-level classification tasks such as single-sentence classification, sentence-pair classification, and machine reading comprehension.

One might wonder how SubChar tokenizers handle character-level tasks where classification is done on every single character, such as sequence labeling and span extraction. Since SubChar tokenizers may combine multiple characters into one token or split one character into sub-character tokens, directly adding a classification head on each token may cause discrepancy with the human annotation, which is done on the character level. For example, it is infeasible to evaluate the POS tag of a sub-character token.

To handle such situations, we perform classification on the character level for these tasks. To obtain the representation of each character, we average the representations of all its sub-character tokens. We apply this on the final layer of BERT and feed the character representation to a linear classifier for downstream tasks.

We measure the performance of this approach on CMRC (span-extraction reading comprehension) and CLUENER (named entity recognition) and show the results in Table 6. The results show

that our model can indeed handle character-level tasks with this simple adaptation. There might be better ways of adopting our model on character-level tasks, and we leave it to future work.

5 Analysis

In this section, we conduct various analyses to better understand the working mechanisms of SubChar tokenization, including illustrations of the efficiency improvement and ablations on different components of our tokenization pipeline.

5.1 Vocabulary Composition

We break down the vocabulary of each tokenizer into three different categories: sub-character tokens, character tokens, and character combination tokens (words and phrases). As shown in Figure 5, character tokenizers only have character tokens, while sub-word tokenizers have a small percentage of combination tokens. The main reason for the relatively small number of combination tokens in sub-word tokenizers is that unlike how English words are composed with 26 alphabet letters, there are thousands of unique Chinese

	iFLYTEK	TNEWS
CharTokenizer	289.0	22.0
Sub-word	255.2	20.1
SubChar-Wubi	183.2	15.8
SubChar-Pinyin	185.2	16.1
SubChar-Pinyin-NoIndex	**175.4**	**15.2**

Table 7: Comparison of average length of tokenized sequences with different tokenizers. SubChar tokenizers produce much shorter tokenized sequences than the baselines. SubChar-Pinyin-NoIndex tokenizer achieves the most length reduction. BPE and Unigram LM counterparts achieve similar speedup improvement.

	TNEWS	iFLYTEK
CharTokenizer	100.0%	100.0%
Sub-word	99.9%	92.6%
SubChar-Wubi	87.0%	69.6%
SubChar-Pinyin	83.8%	70.4%
SubChar-Pinyin-NoIndex	**82.7%**	**68.9%**

Table 8: Finetuning time of models with different tokenizers. Numbers indicate time relative to the CharTokenizer baseline model. Models with Sub-Char tokenizers take much shorter time to finish finetuning. SubChar-Pinyin-NoIndex brings the most speedup.
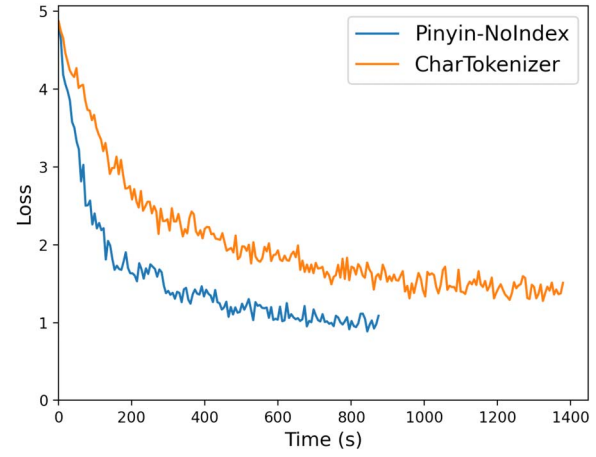
characters, which take up a large proportion of the vocabulary in order to maintain coverage.

In contrast, SubChar tokenizers use a very small fraction of sub-character tokens to compose many complex Chinese characters, therefore saving up a large percentage of the vocabulary to store combination tokens. This brings the advantage of having more words and phrases in the tokenized outputs, thus shortening the sequence lengths, as elaborated in the next section.

5.2 Efficiency Improvement

The direct consequence of having more character combinations in the vocabulary is that the tokenized sequences are shorter. Table 7 shows the average sequence length by using different tokenizers on two downstream datasets. We observe that SubChar tokenizers can tokenize the inputs into much shorter sequences.

Moreover, our SubChar tokenizers can speed up training for both pretraining and finetuning. During finetuning, we can pack multiple sequences into one input sequence to reduce the computation waste introduced by sequence padding (Krell et al., 2021), and shorter sequence lengths allow the sequences to be packed more densely, thus increasing the overall throughput.

Table 8 shows the model finetuning time relative to the CharTokenizer baseline. We observe significant speedup by SubChar tokenizers, finishing in as little as 68.9% time on iFLYTEK with the SubChar-Pinyin-NoIndex tokenizer. In Figure 6, we plot the training curves for the CharTokenizer baseline and the SubChar-Pinyin-NoIndex model on the iFLYTEK dataset; we observe that

Figure 6: Training curves on the iFLYTEK dataset with two different models. The y-axis indicates classification loss (cross-entropy), the x-axis indicates time (seconds). Our SubChar-Pinyin-NoIndex model gets a lower loss than the CharTokenizer baseline throughout training.

our SubChar-Pinyin-Noindex model indeed converges much faster and achieves lower training loss in the end.

The speedup on pretraining is also significant. While the running speed differs on different machines, the compression brought by the shorter tokenized outputs is hardware-invariant. In Table 9, we show the relative size (disk memory) of the tokenized pretraining corpus. We observe that SubChar tokenizers can tokenize the raw pretraining texts into shorter sequences than the baselines, thus resulting in a much smaller pretraining data (e.g., as much as 25.3% smaller than that of the CharTokenizer baseline with SubChar-Pinyin-NoIndex). In turn, this can translate to much faster pretraining on any training infrastructure.

479

	Tokenized Corpus Size
CharTokenizer	100.0%
Sub-word	91.4%
SubChar-Wubi	77.2%
SubChar-Pinyin	78.4%
SubChar-Pinyin-NoIndex	**74.7%**

Table 9: Relative size (disk memory) of the tokenized pretraining corpus with different tokenizers. SubChar tokenizers produce much smaller tokenized corpus due to their ability to tokenize inputs into shorter sequences.

5.3 Impact of Vocabulary Size

Intuitively, when we increase the vocabulary size, there will also be more room to store combination tokens (e.g., words and phrases), leading to a decrease in tokenization length and thus better efficiency. Although we used the standard vocabulary size of 22675 in our previous experiments, to understand whether the efficiency benefits of SubChar tokenization wear off at larger vocabulary size, we perform an additional ablation on the impact of vocabulary size.

As shown in Table 10, as we increase the vocabulary size, the efficiency advantage of SubChar tokenizers slightly diminishes. However, even at a very large vocab size of 60000, our SubChar-Pinyin tokenizer still tokenizes the inputs into significantly shorter sequences than the Sub-word baseline. We thus conclude that the efficiency advantage of our SubChar tokenizers would hold in most practical cases where the vocabulary size is typically under 60000 (such as BERT and RoBERTa).

5.4 Impact of Pretraining Data Size

To understand the impact of pretraining data size, we take the checkpoints of the 12-layer Transformer models pretrained on the 2.3G Baike corpus, and further pretrain them on a much larger corpus of 22.1GB text. This 22.1GB corpus is sampled from Chinese web text,[9] mainly consisting of books and web pages. We further pretrain for 8K steps with a maximum sequence length of 512.

As shown in the bottom block of Table 2, further training on this larger corpus leads to small improvement on average performance (72.23 → 72.81 for CharTokenizer and 72.87 → 73.42 for

[9] https://github.com/OpenBMB/CPM-Live.

	iFLYTEK	TNEWS
Vocab Size = 22675		
Sub-word	255.2	20.1
SubChar-Pinyin-NoIndex	**175.4**	**15.2**
Vocab Size = 40000		
Sub-word	188.9	15.9
SubChar-Pinyin-NoIndex	**166.1**	**14.4**
Vocab Size = 60000		
Sub-word	176.2	14.9
SubChar-Pinyin-NoIndex	**164.0**	**14.1**

Table 10: Comparison of average length of tokenized sequences with different tokenizers and different vocabulary sizes.

SubChar-Pinyin), possibly because the original models trained on 2.3GB corpus are already close to being fully trained. More importantly, this result shows that even with pretraining on larger corpora, our proposed methods can still match or slightly outperform baselines on the downstream datasets.

5.5 Impact of Encoding Methods

As described in Section 2, we experiment with different types of encoding methods and compare their downstream performance to analyze the impact.

Our previous encoding methods are based on the hypothesis that linguistic information such as glyph or pronunciation provides useful inductive biases to the model. However, in the case where this hypothesis is not true, it is possible that non-linguistic encoding methods may work as well. To verify this, we add two encoding methods that do not consider any linguistic information: *Byte Encoding* and *Random Index Encoding*, for the purpose of ablation analysis.

In *Byte Encoding*, we convert every character into its byte sequence, same as in ByT5 (Xue et al., 2022). In cases where the byte sequence consists of multiple indices (each Chinese character has three byte indices), we concatenate them and append the character separation symbol as the encoding (e.g., 魑 → *233_173_145#*).

In *Random Index Encoding*, we map each character into a unique and randomly generated five-digit index and append the character separation symbol as the encoding (e.g., 魑 → *29146#*).

	TNEWS	IFLY	BQ	WSC	AFQMC	CSL	OCNLI	AVG
SubChar-Pinyin	63.68	58.81	81.74	65.90	68.89	82.87	67.98	70.16
SubChar-Zhuyin	64.91	59.39	81.41	62.72	69.14	82.60	69.12	69.90
SubChar-Stroke	64.26	55.44	81.52	62.06	69.88	83.16	68.98	69.33
SubChar-Wubi	63.81	58.74	81.55	64.61	69.66	82.44	68.02	69.90
SubChar-Zhengma	63.86	59.51	81.59	63.27	70.47	82.91	69.03	70.09
SubChar-Cangjie	64.10	57.77	81.98	62.39	68.95	82.60	68.46	69.46
SubChar-Byte	63.58	59.55	81.65	63.60	68.60	82.66	67.93	69.65
SubChar-RandomIndex	64.11	59.16	81.64	63.93	68.53	82.86	69.39	69.95
SubChar-Pinyin (BPE)	63.86	58.84	82.12	65.57	69.86	82.86	68.57	70.24

Table 11: Results of SubChar tokenizers when using different encoding methods. The last row is a model with SubChar-Pinyin tokenizer using BPE as the subword tokenization algorithm, all previous rows are using unigram LM as the subword tokenization implementation. All models have 6 layers with the same hyper-parameters. The impact of different encoding methods on downstream performance is small, and the ULM and BPE versions of SubChar-Pinyin also achieve similar results.

We train SubChar tokenizers with all the different encoding methods and compare the corresponding BERT models using these tokenizers on downstream tasks. The results are presented in Table 11. We observe that the differences between these different tokenizers are rather small in terms of the model performance on downstream datasets. Moreover, perhaps somewhat surprisingly, tokenizers with the non-linguistic encoding methods—SubChar-Byte and SubChar-RandomIndex—can also perform competitively despite the fact that they do not capture glyph or pronunciation information like the other tokenizers.

These results suggest that linguistic encoding may not be necessary for SubChar tokenizers to achieve high performance on downstream tasks. However, the linguistic encoding methods can build more robust and efficient tokenizers as illustrated in previous sections.

5.6 Impact of Vocabulary Construction Algorithm

In previous experiments, we used the Unigram LM implementation in SentencePiece for vocabulary construction. We perform an additional ablation where we replace Unigram LM with BPE for vocabulary construction to train a pinyin-based tokenizer, while holding all other hyperparameters constant.

We compare the SubChar-Pinyin-BPE variant with the unigram LM (SubChar-Pinyin) tokenizer. We find that these two perform similarly.

In terms of **efficiency**: SubChar-Pinyin-BPE tokenizes iFLYTEK to an average length of 184.4 and tokenizes TNEWS to an average length of 15.9. In comparison, SubChar-Pinyin tokenizes iFLY-TEK to an average length of 185.2 and tokenizes TNEWS to an average length of 16.1. The vocabulary compositions of the two are also similar, where character combination takes up the majority of the space in the vocabulary for both BPE and unigram LM implementations. In terms of **performance**, we observe in Table 11 that the BPE implementation and the unigram LM implementation have little difference in downstream task performance. Based on these results, we conclude that the choice of which vocabulary construction to use has a marginal impact on the tokenization efficiency and model performance.

6 Related Work

Chinese PLMs. Chinese BERT (Devlin et al., 2019) is the first Chinese PLM, and it adopts character tokenization. Since then, researchers have explored techniques to explicitly incorporate the word-level information into Chinese PLMs for better performance. Zhu (2020) and Zhang et al. (2021a) expand BERT vocabulary with Chinese words apart from Chinese characters and incorporate them in the pretraining objectives. Cui et al. (2019a), Wei et al. (2019), and Xiao et al. (2021) consider coarse-grained information through masking whole words and n-grams

481

during the masked language modeling pretraining. Diao et al. (2020) incorporate word-level information via superimposing the character and word embeddings. Lai et al. (2021) incorporate Chinese word lattice structures in pretraining. Different from these studies, we investigate the information in the sub-character level for Chinese PLMs.

Linguistically Informed Techniques for Chinese NLP. Before the era of PLM, many efforts have been made to incorporate linguistic knowledge, including both glyph (Sun et al., 2014; Yu et al., 2017; Cao et al., 2018) and pronunciation (Zhang et al., 2019b; Chaudhary et al., 2018), into word embedding (Mikolov et al., 2013). Beyond word-level representation, researchers explore the use of linguistic information to enhance sequential models (Dong et al., 2016; Bharadwaj et al., 2016; Liu et al., 2017), especially BERT (Meng et al., 2019; Sun et al., 2021). Compared to these works, we do not incorporate additional information from sources like images, instead, our proposed tokenization methods are drop-in replacements to existing tokenizers, without adding any extra layers or parameters. Besides, cws is a common preprocessing step for Chinese NLP (Li and Sun, 2009); Li et al. (2019) empirically analyze whether cws is helpful for Chinese NLP tasks before the era of PLMs and find that the answer is no in many cases. In our work, we also spend a section examining the impact of cws specifically for PLMs. Moreover, as shown by Huang et al. (2021), incorporating linguistic information also benefits spelling check. Instead of explicitly using spelling check, our linguistically informed tokenizations are robust to spelling errors.

Granularity of Tokenization. Although subwords are taken to be the default granularity of tokenization since the release of BERT, researchers also explore different granularities for PLMs. For instance, ELMo (Peters et al., 2018), the early pioneer of PLMs, starts by using character representation. Ma et al. (2020) combine character representations with sub-word representations for better performance and robustness. Nzeyimana and Rubungo (2022) incorporate a morphological analyzer for tokenization and achieve gains for the Kinyarwanda language model. More recently, there has been a trend in tokenization-free methods, including Byte-BPE (Wei et al., 2021), CANINE (Clark et al., 2021), ByT5 (Xue et al., 2022), and Charformer (Tay et al., 2022), which discard explicit tokenization and directly represent inputs as small units such as bytes. The downside of these tokenization-free approaches is obvious: The longer tokenized sequence lengths slow down both training and inference. Contrary to these, our sub-character tokenization encourages the use of more character combinations, which largely shortens the tokenized sequences.

7   Conclusion

In this work, we propose sub-character tokenization and conduct comprehensive experiments to illustrate its advantage over existing tokenization methods. Compared to treating each individual character as a token (CharTokenizer) or directly running sub-word tokenization on the raw Chinese text (sub-word tokenizer), our SubChar tokenizers not only perform competitively on downstream NLU tasks, but, more importantly, they can be much more efficient and robust. We conduct a series of ablation and analysis to understand the reasons why SubChar tokenizers are more efficient, as well as the impact of linguistic and non-linguistic encoding. Given the advantages of our SubChar tokenizers, we believe that they are better alternatives to all existing Chinese tokenizers, especially in applications where efficiency and robustness are critical. It is possible that our approach can be useful for other morphologically poor languages and more complicated methods could be developed based on SubChar tokenization for even better performance. We leave these interesting directions for future exploration. On a broader level, our work makes an important attempt in developing more tailored methods for a language drastically different from English with promising results. We believe that this is a crucial future direction for the community given the language diversity in the world. We hope that our work can inspire more such work in order to benefit language technology users from different countries and cultures.

Limitations

Our experiments are focused on natural language understanding tasks. We recognize that adapting our SubChar tokenization to language generation

tasks might require additional efforts, for example, we may want to avoid cases of predicting sub-character tokens that do not form complete characters. Also, evaluating the robustness of language generation models on real-world input noises may require additional benchmarks beyond those used in this paper. We leave such exploration as an interesting direction for future work.

Another limitation is that our method is designed specifically for the Chinese language. While we hypothesize that our method can also bring benefits to other languages with ideographic symbols, such as Kanji in Japanese, we leave such investigation to future work.

Broader Impact

We expect our work to have a positive impact on the society. Firstly, we addressed the practical problem of handling input with real-world noises. Such noisy settings are very common in real-life applications. Our method, along with the evaluation framework, can help make language technologies more robust and reliable in real-world applications, especially for Chinese users. Secondly, we addressed the efficiency concern of large language models by significantly reducing both and training and inference time. This not only reduces the latency of these models in real-world applications, but, more importantly, helps reduce the environmental costs of using these large language models, moving further towards Green AI. All of our code and models are released with proper documentation in order to better facilitate the adoption of our work in a wide range of research and industrial applications.

Acknowledgments

Author contributions Chenglei Si, Zhengyan Zhang, and Yingfa Chen wrote the code and conducted the experiments. Chenglei was in charge of tokenzer training and pretraining experiments, Zhengyan did the CWS experiments, Yingfa did the finetuning experiments. All three contributed to the analysis experiments. Chenglei Si, Zhengyan Zhang, and Yingfa Chen wrote the initial draft; Fanchao Qi, Xiaozhi Wang, and Zhiyuan Liu significantly edited and improved the paper. Yasheng Wang, Qun Liu, and Maosong Sun provided valuable advice to the research. Chenglei started this work back when he was visiting the THUNLP group in 2021.

References

Emily Bender. 2019. The #BenderRule: On naming the languages we study and why it matters. *The Gradient*.

Akash Bharadwaj, David Mortensen, Chris Dyer, and Jaime Carbonell. 2016. Phonologically aware neural model for named entity recognition in low resource transfer settings. In *Proceedings of EMNLP*, pages 1462–1472. https://doi.org/10.18653/v1/D16-1153

Shaosheng Cao, Wei Lu, Jun Zhou, and Xiaolong Li. 2018. cw2vec: Learning chinese word embeddings with stroke n-gram information. In *Proceedings of AAAI*.

Pi-Chuan Chang, Michel Galley, and Christopher D. Manning. 2008. Optimizing Chinese word segmentation for machine translation performance. In *Proceedings of the Third Workshop on Statistical Machine Translation*. https://doi.org/10.3115/1626394.1626430

Aditi Chaudhary, Chunting Zhou, Lori Levin, Graham Neubig, David R. Mortensen, and Jaime Carbonell. 2018. Adapting word embeddings to new languages with morphological and phonological subword representations. In *Proceedings of EMNLP*, pages 3285–3295. https://doi.org/10.18653/v1/D18-1366

Jing Chen, Qingcai Chen, Xin Liu, Haijun Yang, Daohe Lu, and Buzhou Tang. 2018. The BQ corpus: A large-scale domain-specific Chinese corpus for sentence semantic equivalence identification. In *Proceedings of EMNLP*. `https://doi.org/10.18653/v1/D18 -1536`

Jonathan Clark, Dan Garrette, Iulia Turc, and John Wieting. 2021. CANINE: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91. `https://doi.org/10.1162/tacl_a_00448`

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *Proceedings of ICLR*.

Florian Coulmas. 1991. *The Writing Systems of the World*. Blackwell Publishers.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for Chinese natural language processing. In *Findings of EMNLP*.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019a. Pre-training with whole word masking for Chinese BERT. *IEEE/ACM TASLP*, 29:3504–3514.

Yiming Cui, Ting Liu, Wanxiang Che, Li Xiao, Zhipeng Chen, Wentao Ma, Shijin Wang, and Guoping Hu. 2019b. A span-extraction dataset for Chinese machine reading comprehension. In *Proceedings of EMNLP-IJCNLP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*.

Shizhe Diao, Jiaxin Bai, Yan Song, Tong Zhang, and Yonggang Wang. 2020. ZEN: Pre-training Chinese text encoder enhanced by n-gram representations. In *Findings of EMNLP*. `https://doi.org/10.18653/v1/2020 .findings-emnlp.425`

Chuanhai Dong, Jiajun Zhang, Chengqing Zong, Masanori Hattori, and Hui Di. 2016. Character-based lstm-crf with radical-level features for chinese named entity recognition. In *International Conference on Computer Pro-*

cessing of Oriental Languages. `https:// doi.org/10.1007/978-3-319-50496 -4_20`

Yen-Chen Hao and Chung-Lin Martin Yang. 2021. The effect of second-language orthographic input on the phonological encoding of Mandarin words. *Applied Psycholinguistics*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *Proceedings of ICLR*.

Hai Hu, Kyle Richardson, Liang Xu, Lu Li, Sandra Kübler, and Lawrence Moss. 2020. OCNLI: Original Chinese natural language inference. In *Findings of EMNLP*. `https:// doi.org/10.18653/v1/2020.findings -emnlp.314`

Li Huang, Junjie Li, Weiwei Jiang, Zhiyu Zhang, Minchuan Chen, Shaojun Wang, and Jing Xiao. 2021. PHMOSpell: Phonological and morphological knowledge guided Chinese spelling check. In *Proceedings of ACL*, pages 5958–5967. `https://doi.org/10 .18653/v1/2021.acl-long.464`

Mario Michael Krell, Matej Kosec, Sergio P. Perez, and Andrew Fitzgibbon. 2021. Efficient sequence packing without cross-contamination: Accelerating large language models without impacting performance. *arXiv preprint*, abs/ 2107.02027.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of ACL*. `https://doi.org/10.18653 /v1/P18-1007`

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of EMNLP System Demonstrations*. `https://doi.org/10 .18653/v1/D18-2012`

Yuxuan Lai, Yijia Liu, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2021. LatticeBERT: Leveraging multi-granularity representations in Chinese pre-trained language models. In *Proceedings of NAACL-HLT*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and

Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proceedings of ICLR*.

Hector Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Jingyang Li and Maosong Sun. 2007. Scalable term selection for text categorization. In *Proceedings of EMNLP*.

Xiaoya Li, Yuxian Meng, Xiaofei Sun, Qinghong Han, Arianna Yuan, and Jiwei Li. 2019. Is word segmentation necessary for deep learning of Chinese representations? In *Proceedings of ACL*.

Zhongguo Li and Maosong Sun. 2009. Punctuation as implicit annotations for Chinese word segmentation. *Computational Linguistics*.

Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. 2017. Learning character-level compositionality with visual features. In *Proceedings of ACL*, pages 2059–2068.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint*, abs/1907.11692.

Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. 2020. CharBERT: Character-aware pre-trained language model. In *Proceedings of COLING*.

R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of ACL*. https://doi.org/10.18653/v1/P19-1334

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. 2019. Glyce: Glyph-vectors for Chinese character representations. In *Proceedings of NeurIPS*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, volume 26.

Antoine Nzeyimana and Andre Niyongabo Rubungo. 2022. KinyaBERT: A morphology-aware kinyarwanda language model. In *Proceedings of ACL*. https://doi.org/10.18653/v1/2022.acl-long.367

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*. https://doi.org/10.18653/v1/N18-1202

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. https://doi.org/10.1109/ICASSP.2012.6289079

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of ACL*. https://doi.org/10.18653/v1/P16-1162

Chenglei Si, Zhengyan Zhang, Yingfa Chen, Xiaozhi Wang, Zhiyuan Liu, and Maosong Sun. 2023. READIN: A Chinese multi-task benchmark with realistic and diverse input noises. *arXiv*, abs/2302.07324.

Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. 2020. Investigating prior knowledge for challenging Chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*. https://doi.org/10.1162/tacl_a_00305

Maosong Sun, Xinxiong Chen, Kaixu Zhang, Zhipeng Guo, and Zhiyuan Liu. 2016. THULAC: An efficient lexical analyzer for Chinese. *GitHub*.

Yaming Sun, Lei Lin, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. 2014. Radical-enhanced chinese character embedding. In *Proceedings of COLING*, pages 279–286.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Xuyi Chen, Han Zhang, Xin Tian, Danxiang Zhu, Hao Tian, and Hua Wu. 2019. ERNIE: Enhanced representation through knowledge integration. *arXiv preprint*, abs/1904.09223.

Zijun Sun, Xiaoya Li, Xiaofei Sun, Yuxian Meng, Xiang Ao, Qing He, Fei Wu, and Jiwei

Li. 2021. ChineseBERT: Chinese pretraining enhanced by glyph and Pinyin information. In *Proceedings of ACL*, pages 2065–2075. https://doi.org/10.18653/v1/2021 .acl-long.161

Yi Tay, Vinh Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *Proceedings of ICLR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*.

Junqiu Wei, Qun Liu, Yinpeng Guo, and Xin Jiang. 2021. Training multilingual pre-trained language model with byte-level │ subwords. *arXiv preprint*, abs/2101.09469.

Junqiu Wei, Xiaozhe Ren, Xiaoguang Li, Wenyong Huang, Yi Liao, Yasheng Wang, Jiashu Lin, Xin Jiang, Xiao Chen, and Qun Liu. 2019. NEZHA: Neural contextualized representation for Chinese language understanding. *arXiv*, abs/1904.00204.

Dongling Xiao, Yu-Kun Li, Han Zhang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE-Gram: Pre-training with explicitly n-gram masked language modeling for natural language understanding. In *Proceedings of NAACL-HLT*. https://doi.org/10.18653 /v1/2021.naacl-main.136

Liang Xu, Qianqian Dong, Cong Yu, Yin Tian, Weitang Liu, Lu Li, and Xuanwei Zhang. 2020a. CLUENER2020: Fine-grained name entity recognition for Chinese. *arXiv preprint*, abs/2001.04351.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, Yin Tian, Qianqian Dong, Weitang Liu, Bo Shi, Yiming Cui, Junyi Li, Jun Zeng, Rongzhao Wang, Weijian Xie, Yanting Li, Yina Patterson, Zuoyu Tian, Yiwen Zhang, He Zhou, Shaoweihua Liu, Zhe Zhao, Qipeng Zhao, Cong Yue, Xinrui Zhang, Zhengliang Yang, Kyle Richardson, and Zhenzhong Lan. 2020b. CLUE: A Chinese language understanding evaluation benchmark. In *Proceedings of COLING*.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*.

Jinxing Yu, Xun Jian, Hao Xin, and Yangqiu Song. 2017. Joint embeddings of Chinese words, characters, and fine-grained subcharacter components. In *Proceedings of EMNLP*, pages 286–291.

Xinsong Zhang, Pengshuai Li, and Hang Li. 2021a. AMBERT: A pre-trained language model with multi-grained tokenization. In *Findings of ACL*. https://doi.org/10 .18653/v1/2021.findings-acl.37

Yuan Zhang, Jason Baldridge, and Luheng He. 2019a. PAWS: Paraphrase adversaries from word scrambling. In *Proceedings of NAACL-HLT*.

Yun Zhang, Yongguo Liu, Jiajing Zhu, Ziqiang Zheng, Xiaofeng Liu, Weiguang Wang, Zijie Chen, and Shuangqing Zhai. 2019b. Learning chinese word embeddings from stroke, structure and pinyin of characters. In *Proceedings of CIKM*, pages 1011–1020. https://doi .org/10.1145/3357384.3358005

Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan, Zhiyuan Liu, Minlie Huang, Wentao Han, Yang Liu, Xiaoyan Zhu, and Maosong Sun. 2021b. CPM-2: Large-scale cost-effective pre-trained language models. *arXiv preprint*, abs/2106 .10715. https://doi.org/10.1016/j .aiopen.2021.12.003

Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, Fanchao Qi, Xiaozhi Wang, Yanan Zheng, Guoyang Zeng, Huanqi Cao, Shengqi Chen, Daixuan Li, Zhenbo Sun, Zhiyuan Liu, Minlie Huang, Wentao Han, Jie Tang, Juanzi Li, Xiaoyan Zhu, and Maosong Sun. 2020. CPM: A large-scale generative Chinese pre-trained language model.

AI Open. `https://doi.org/10.1016/j.aiopen.2021.07.001`

Chujie Zheng, Minlie Huang, and Aixin Sun. 2019. ChID: A large-scale Chinese IDiom dataset for cloze test. In *Proceedings of ACL.*

`https://doi.org/10.18653/v1/P19-1075`

Wei Zhu. 2020. MVP-BERT: Redesigning vocabularies for Chinese BERT and multi-vocab pretraining. *arXiv preprint*, abs/2011.08539.