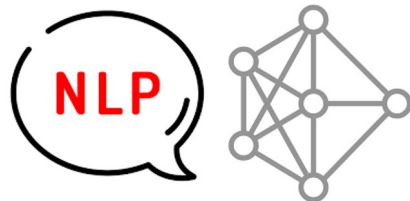




CHULA ENGINEERING  
Foundation toward Innovation

COMPUTER



## Subword

2110572: Natural Language Processing Systems

Assoc. Prof. Peerapon Vateekul, Ph.D.

Department of Computer Engineering,  
Faculty of Engineering, Chulalongkorn University  
[peerapon.v@chula.ac.th](mailto:peerapon.v@chula.ac.th)

Credits to: Aj.Ekapol & TA team (TA.Pluem, TA.Knight, and all TA alumni)



# Introduction

Problem:

- 1) out-of-vocab
- 2) large vocabulary size

Solution: subword embedding

- 1) Byte-Pair Encoding (BPE)
- 2) WordPiece
- 3) Unigram
- 4) Sentencepiece





+

Byte-Pair Encoding (BPE)



## Byte-Pair Encoding (BPE)

4

BPE was introduced in Neural Machine Translation of Rare Words with Subword Units ([Sennrich et al., 2015](#)).

Used in [GPT-2](#), [Roberta](#), and even [ChatGPT](#)

Relies on a pre-tokenizer that splits the training data into words.

Next, BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns to merge rules to form a new symbol from two symbols of the base vocabulary ([similar to huffman coding; frequencies](#)).



## BPE example (1 sentence)

- aaabdaaaba

- **Z**abd**Z**abac

- Z=aa

- **Z****Y**d**Z****Y**ac

- Y=ab

- Z=aa

- **X**d**X**ac

- X=ZY

- Y=ab

- Z=aa

- พราร และ ขาร นั้ง บน ราร ดู ข่าว กราร บน คาร

- พร**X** และ ข**X** นั้ง บน ร**X** ดู ข**X** กร**X** บน ค**X**

- X=าร

- พ**y** และ ข**X** นั้ง บน **y** ดู ข**X** ค**y** บน ค**X**

- X=าร

- y=ร**X**

- พ**y** และ ข**X** นั้ง **z y** ดู ข**X** ค**y** **z** ค**X**

- X=าร

- y=ร**X**

- z=บน

## + BPE - training

6

### Example corpus

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

The most frequent symbol pair is "u" followed by "g", occurring  $10 + 5 + 5 = 20$  times in total. Thus, the first merge rule the tokenizer learns is to group all "u" symbols followed by a "g" symbol together. Next, "ug" is added to the vocabulary.

```
("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

# + BPE - usage

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

7

## Tokenization algorithm

Tokenization follows the training process closely, in the sense that new inputs are tokenized by applying the following steps:

1. Normalization
2. Pre-tokenization
3. Splitting the words into individual characters
4. Applying the merge rules learned in order on those splits

Let's take the example we used during training, with the three merge rules learned:

```
("u", "g") -> "ug"  
("u", "n") -> "un"  
("h", "ug") -> "hug"
```

### How to use

- bug = ["b", "ug"] ("b" in dict)
- mug = ["UNK", "ug"] ("m" not in dict)
- thug = ["UNK", "hug"] ("t" not in dict)



Wordpiece





# WordPiece

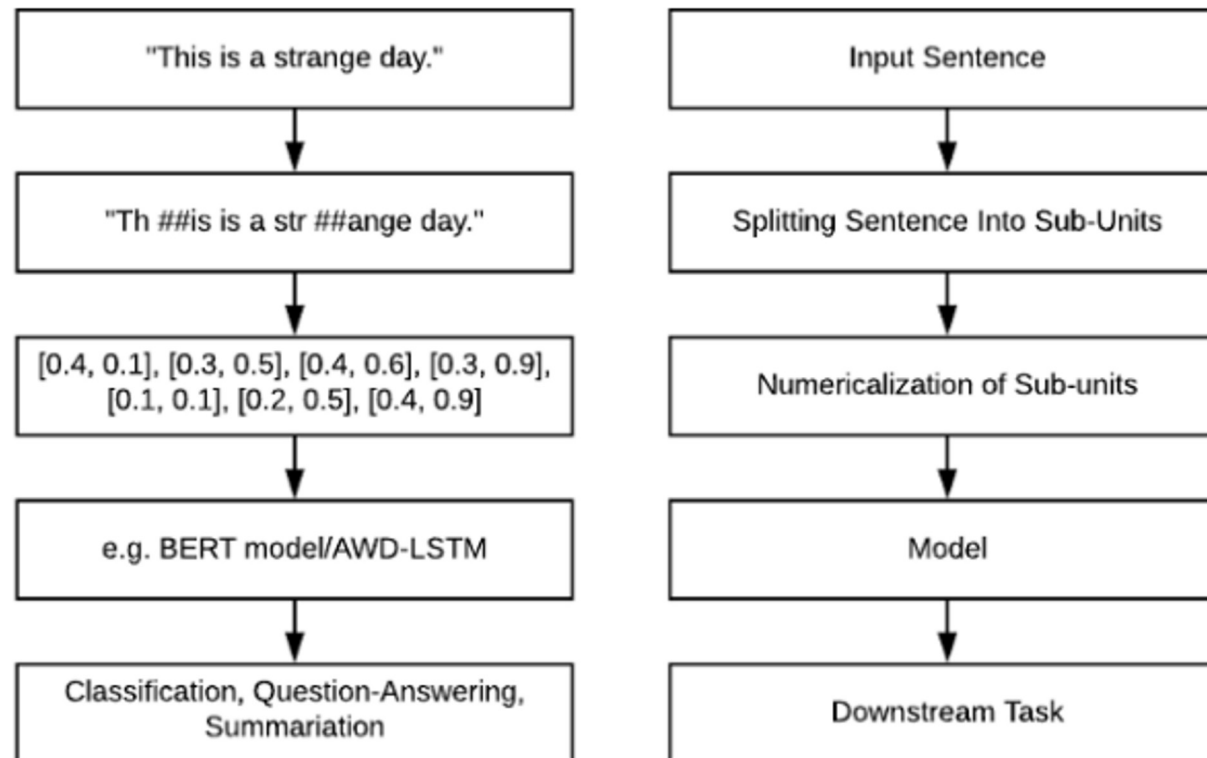
- Google NMT(GNMT) uses a variant of this
  - V1: wordpiece model
  - V2: sentencepiece model
- WordPiece is the subword tokenization algorithm used for models such as [BERT](#), [DistilBERT](#), and [Electra](#).
- Rather than char n-gram count, uses a greedy approximation to maximize language model log likelihood to choose the pieces (add n-gram that maximally reduces perplexity)
- Like BPE, WordPiece learns merge rules. The main difference is the way the pair to be merged is selected. [Instead of selecting the most frequent pair, WordPiece computes a score for each pair using the following formula:](#)

$$\text{score} = (\text{freq\_of\_pair}) / (\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$$



## WordPiece (cont.)

- There are 2 types of tokens: start token (**not** ##), and continuing token (##)





# WordPiece - training

## Corpus

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

The splits here will be:

```
("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g", 5)
```

so the initial vocabulary will be ["b", "h", "p", "##g", "##n", "##s", "##u"]



# WordPiece - training (cont.)

12

## Corpus

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

From initial vocab ["b", "h", "p", "##g", "##n", "##s", "##u"]

the best score goes to the pair ("##g", "##s") — the only one without a "##u" —  $= 5 / (20 * 5) = 1 / 20$ , and the first merge learned is ("##g", "##s") -> ("##gs")

Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]

Corpus: ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)

## + WordPiece - usage

Tokenization differs in WordPiece and BPE in that WordPiece only saves the final vocabulary, not the merge rules learned.

```
Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]
```

How to use: “the longest subword”

- hugs = [“hug”, “##s”]

If not possible to find subwords, tokenize the whole word as UNK.

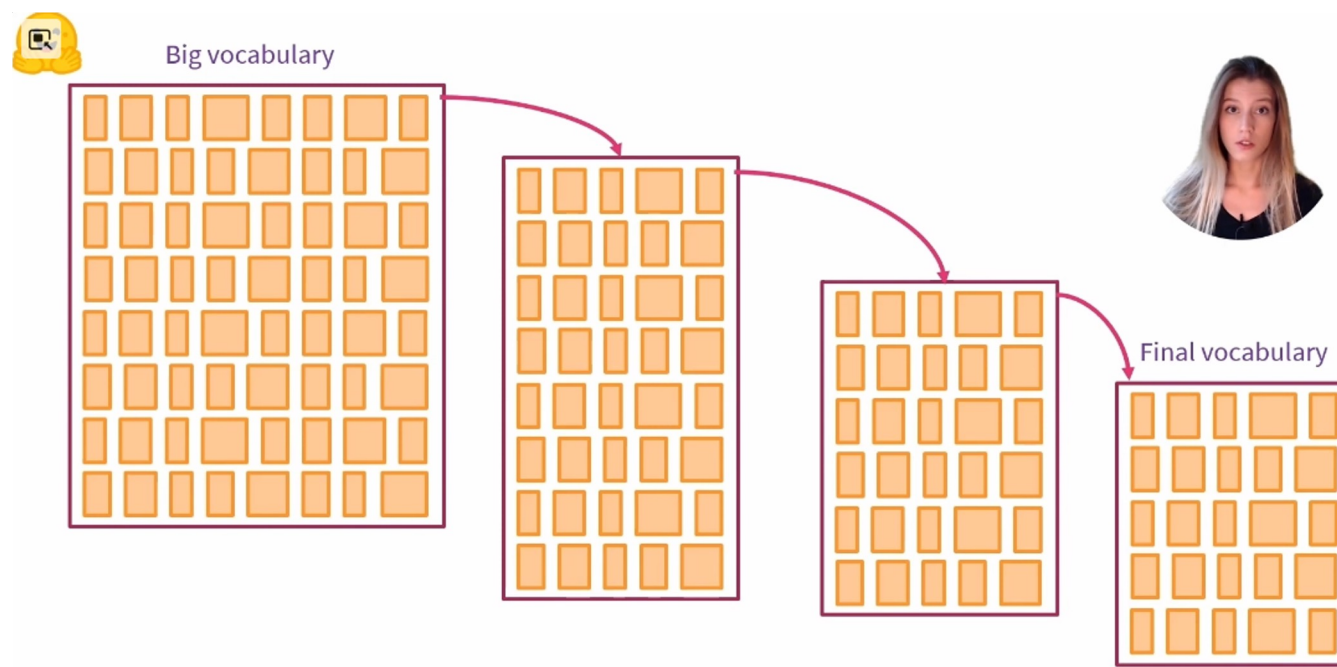
- mug = [“UNK”]
- bum = [“UNK”] ~~(not [“b”, “##u”, UNK])~~

+

Unigram

# + Unigram

Start with a big vocab and reduce it based on a unigram LM loss



## + Unigram - training

17

Initial vocab = **all** substring of corpus

### Corpus

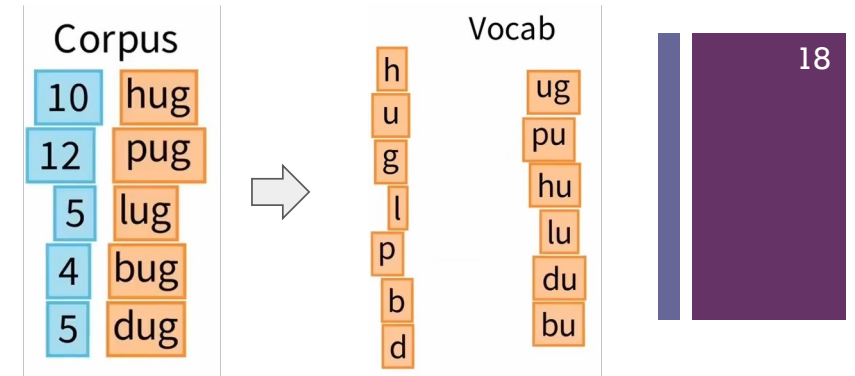
10	hug
12	pug
5	lug
4	bug
5	dug

### Vocab

h	ug
u	pu
g	hu
l	lu
p	du
b	bu
d	



# + Unigram - training (cont.)



1st iteration of EM

**The E step.** Select the split for each word in the corpus with the highest prob.

Vocab

h	10/180	ug	36/180
u	36/180	pu	12/180
g	36/180	hu	10/180
l	5/180	lu	5/180
p	12/180	du	5/180
b	4/180	bu	4/180
d	5/180		

Possible splits for "hug"

$$h \ u \ g \quad \frac{10}{180} \times \frac{36}{180} \times \frac{36}{180} = 2.22e - 03$$

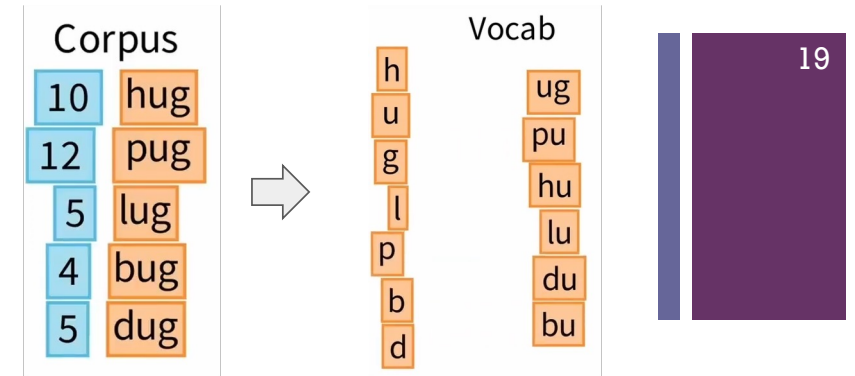
Choose 1 randomly

$$hu \ g \quad \frac{10}{180} \times \frac{36}{180} = 1.11e - 02$$

$$h \ ug \quad \frac{10}{180} \times \frac{36}{180} = 1.11e - 02$$

$$hug \ 0$$

# + Unigram - training (cont.)

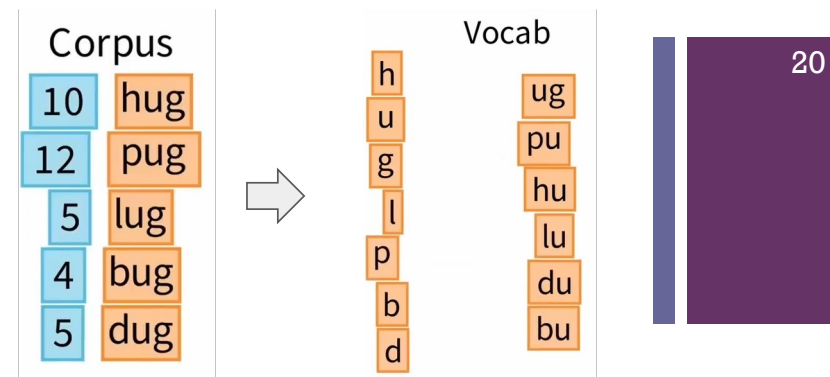


1st iteration of EM

The E step. Calculate loss.

Loss				
Corpus	Splits	Scores	$\sum freq \times (-\log(P(word)))$	
10 hug	→ hu g	1.11e-02	$10 \times (-\log(1.11e - 02))$	
12 pug	→ pu g	1.33e-02	$+12 \times (-\log(1.33e - 02))$	
5 lug	→ lu g	5.56e-03	$+5 \times (-\log(5.56e - 03))$	
4 bug	→ bu g	4.44e-03	$+4 \times (-\log(4.44e - 03))$	
5 dug	→ du g	5.56e-03	$+5 \times (-\log(5.56e - 03))$	
			170.40	

# + Unigram - training (cont.)



1st iteration of EM

**The M step** Remove the tokens that least impact the loss (remove p% at a time)

Try removing **ug**

Vocab

h	10/180	<del>ug</del>	<del>36/180</del>
u	36/180	pu	12/180
g	36/180	hu	10/180
l	5/180	lu	5/180
p	12/180	du	5/180
b	4/180	bu	4/180
d	5/180		

Loss is still the same

Corpus

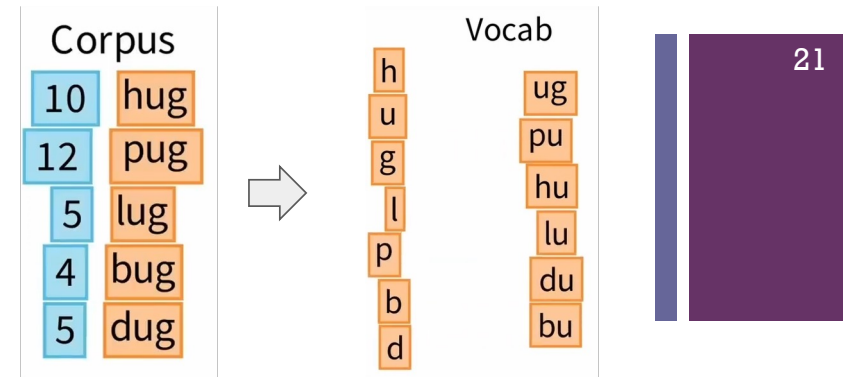
10	hug	→	hu	g	1.11e-02
12	pug	→	pu	g	1.33e-02
5	lug	→	lu	g	5.56e-03
4	bug	→	bu	g	4.44e-03
5	dug	→	du	g	5.56e-03

Removing any token results in the same loss so choose randomly again

	Loss
With all vocabulary	170.4
Without	<del>170.4</del>
	170.4
	170.4
	170.4
	170.4
	170.4
	170.4

Loss 170.40

## + Unigram - training (cont.)



2nd iteration of EM

**The E step.** Select the split for each word in the corpus with highest prob.

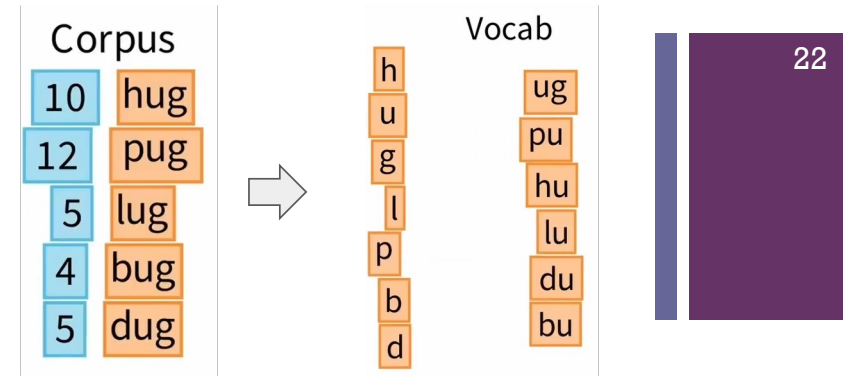
Vocab

h	10/144	pu	12/144
u	36/144	hu	10/144
g	36/144	lu	5/144
l	5/144	du	5/144
p	12/144	bu	4/144
b	4/144		
d	5/144		

Possible splits for "hug"

h	u	g	$\frac{10}{144} \times \frac{36}{144} \times \frac{36}{144} = 4.34e-03$
hu	g	$(10/144) * (36/144) = 1.7e-02$	
h	ug	$\frac{10}{144} \times 0 = 0.00e+00$	
hug	0	$= 0.00e+00$	

# + Unigram - training (cont.)



2nd iteration of EM

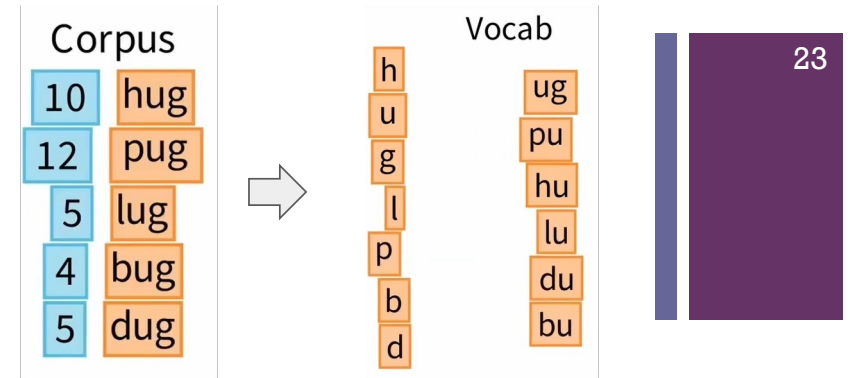
The E step. Calculate loss.

Vocab

h	10/144
u	36/144
g	36/144
l	5/144
p	12/144
b	4/144
d	5/144
pu	12/144
hu	10/144
lu	5/144
du	5/144
bu	4/144

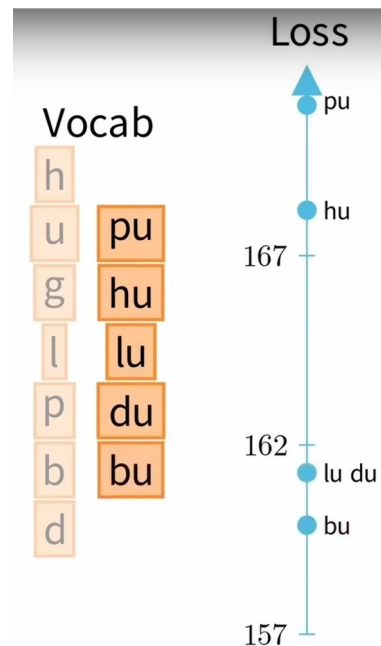
Corpus	Splits	Scores	Loss
10 hug	→ hu g	1.7e-02	168.20
12 pug	→ pu g	2.08e-02	
5 lug	→ lu g	8.68e-03	
4 bug	→ bu g	6.94e-03	
5 dug	→ du g	8.68e-03	

## + Unigram - training (cont.)



2nd iteration of EM

**The M step.** Remove the tokens that least impact the loss (remove p% at a time)



Removing “bu” gives the least loss so “bu” is removed

## + Stop?

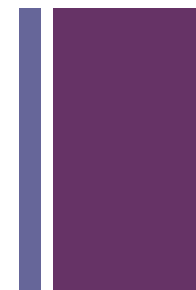
1. Convergence of Likelihood: the likelihood of the data (given the current subword vocabulary) between consecutive iterations is smaller than a predefined threshold
2. Minimal Vocabulary Updates: the changes in the subword vocabulary between iterations become negligible.
3. Maximum Iterations
4. Fixed Vocabulary Size
5. Performance Metrics: Stop when the metric (e.g., perplexity) improvement between iterations plateaus.

+

SentencePiece



# + SentencePiece



- SentencePiece: A simple and language-independent subword tokenizer and detokenizer for Neural Text Processing (Kudo et al., 2018)
- It aims to solve 2 issues.
- Issue 1: Which one should be the correct detokenization?
  - `Tokenize("World.") == Tokenize("World. ")`
- Issue 2: End-to-End to avoid the need of language-specific tokenization.

**WangchanBERTa** We name our pretrained language models according to their architectures, tokenizers and the datasets on which they are trained on. The models can be found on HuggingFace<sup>12</sup>.

	Architecture	Dataset	Tokenizer
wangchanberta-base-wiki-spm	RoBERTa-base	Wikipedia-only	SentencePiece
wangchanberta-base-wiki-newmm	RoBERTa-base	Wikipedia-only	word (newmm)
wangchanberta-base-wiki-ssg	RoBERTa-base	Wikipedia-only	syllable (ssg)
wangchanberta-base-wiki-sefr	RoBERTa-base	Wikipedia-only	SEFR
wangchanberta-base-att-spm-uncased	RoBERTa-base	Assorted Thai Texts	SentencePiece

Table 3: WangchanBERTa model names



## SentencePiece (cont.)

27

Introduces “\_ (U+2581)” to preserve whitespace for detokenization

For the sake of clarity, SentencePiece first escapes the whitespace with a meta symbol \_ (U+2581), and tokenizes the input into an arbitrary subword sequence, for example:

- **Raw text:** Hello\_world.
- **Tokenized:** [Hello] [\_wor] [ld] [.]

As the whitespace is preserved in the tokenized text, we can detokenize the tokens without any ambiguities with the following Python code.

```
detok = ''.join(tokens).replace('_', ' ')
```

Feature	SentencePiece
Supported algorithm	BPE, unigram, char, word

<https://github.com/google/sentencepiece>