Word-based tokenization

'You are learning tokenization'

'You' 'are' 'learning' 'tokenization'

Divides text into individual words

Preserves semantic meaning

Increases the model's overall vocabulary

Character-based tokenization

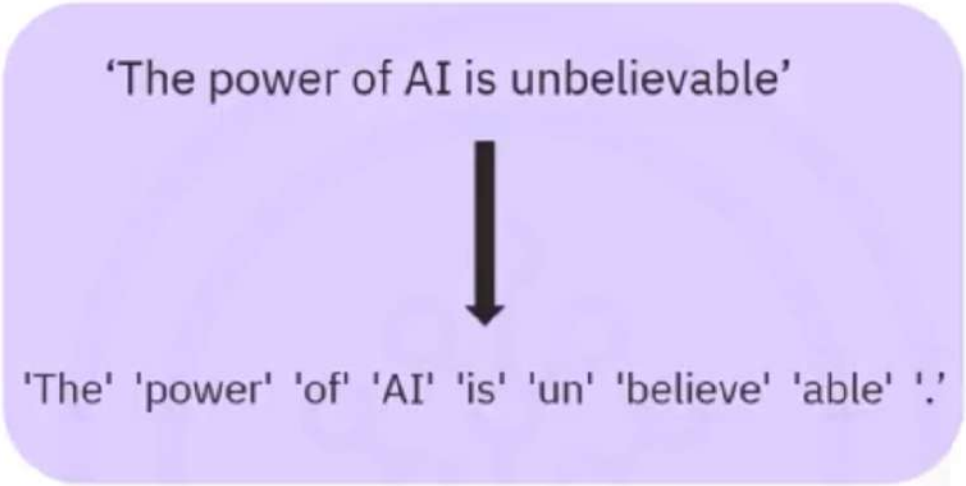
Splits text into individual characters

Smaller vocabularies

May not convey the same information as entire words

Increases input dimensionality and computational needs

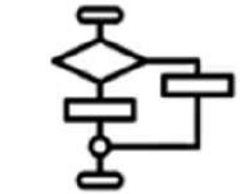
Subword-based tokenization



Frequently used words unsplit;
infrequent words broken down

Combines the advantages of word-based and character-based tokenization

Subword-based tokenization



Algorithms

- **WordPiece** → Evaluates the benefits and drawbacks of splitting and merging two symbols
- **Unigram** → Breaks text into smaller pieces; Narrows down a large list of possibilities based on the frequency of appearance
- **SentencePiece** → Segments text into manageable parts and assigns unique IDs

Subword-Based Tokenizer

WordPiece

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
tokenizer.tokenize("IBM taught me tokenization.")
```

```
Output -
['ibm', 'taught', 'me', 'token', '##ization', '.']
```

Subword-Based Tokenizer

Unigram & SentencePiece

```
from transformers import XLNetTokenizer

tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")
tokenizer.tokenize("IBM taught me tokenization.")
```

```
Output -
['_IBM', '_taught', '_me', '_token', 'ization', '.']
```

Tokenization and indexing in PyTorch

Use torchtext library for tokenization

Use the build_vocab_from_iterator function

Creates a vocabulary from the tokens

NLP	to	of	for	with	of	using

Assigns each token a unique index

1	19	15	13	9	15	20

Tokenization with PyTorch

```
dataset = [
(1,"Introduction to NLP"),
(2,"Basics of PyTorch"),
(1,"NLP Techniques for Text Classification"),
(3,"Named Entity Recognition with PyTorch"),
(3,"Sentiment Analysis using PyTorch"),
(3,"Machine Translation with PyTorch"),
(1," NLP Named Entity,Sentiment Analysis,Machine Translation "),
(1," Machine Translation with NLP "),
(1," Named Entity vs Sentiment Analysis  NLP ")]
```

Tokenization with PyTorch

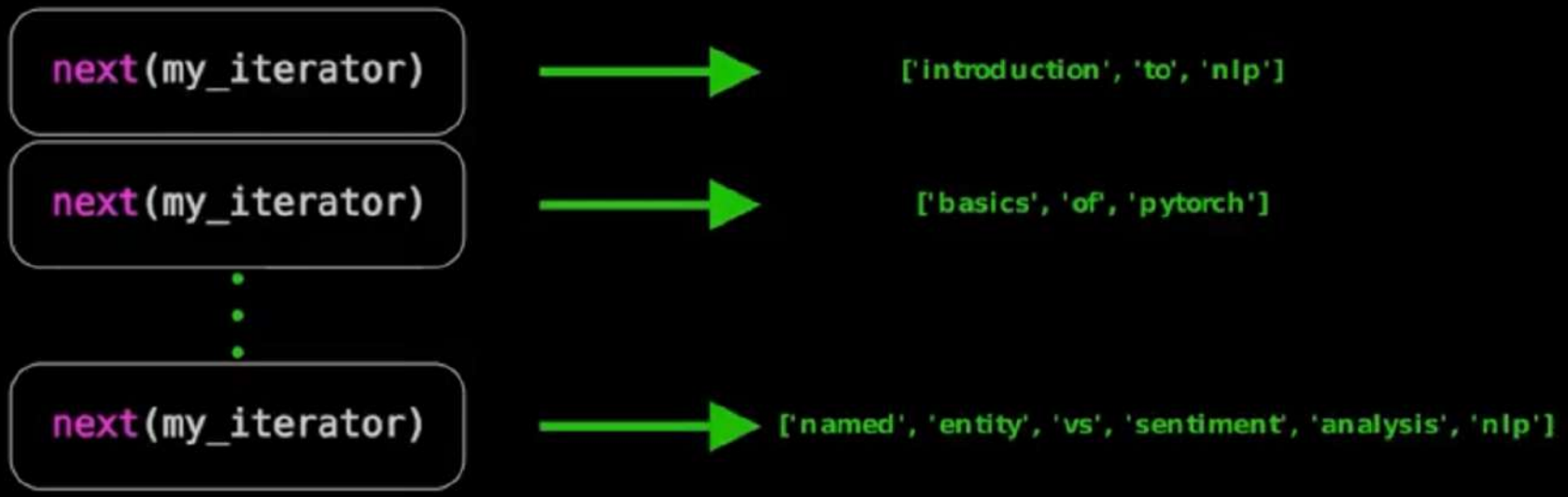
```
from torchtext.data.utils import get_tokenizer

tokenizer = get_tokenizer("basic_english")
tokenizer(dataset[0][1])
```

Tokenization with PyTorch

```
def yield_tokens(data_iter):
    for _,text in data_iter:
        yield tokenizer(text)

my_iterator = yield_tokens(dataset)
```



Tokenization with PyTorch

```
vocab = build_vocab_from_iterator(yield_tokens(dataset), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])
vocab.get_stoi()
```

```
{
'vs': 21,
'to': 19,
'recognition': 16,
'introduction': 14,
'for': 13,
'basics': 11,
'with': 9,
'translation': 8,
'sentiment': 7,
'classification': 12,
'nlp': 1
}
```

```
{
',': 10,
'named': 6,
'using': 20,
'machine': 5,
'text': 18,
'entity': 4,
'techniques': 17,
'<unk>': 0,
'of': 15,
'analysis': 3,
'pytorch': 2
}
```

Tokenization with PyTorch

```
vocab = build_vocab_from_iterator(yield_tokens(dataset), specials=["<unk>"])
vocab.set_default_index(vocab["<unk>"])
vocab.get_stoi()
```

```
{
'vs': 21,
'to': 19,
'recognition': 16,
'introduction': 14,
'for': 13,
'basics': 11,
'with': 9,
'translation': 8,
'sentiment': 7,
'classification': 12,
'nlp': 1
}
```

```
{
',': 10,
'named': 6,
'using': 20,
'machine': 5,
'text': 18,
'entity': 4,
'techniques': 17,
'<unk>': 0,
'of': 15,
'analysis': 3,
'pytorch': 2
}
```

Applying the vocab function

```
vocab(['introduction', 'to', 'nlp'])

[14, 19, 1]
```

Creating tokens and indices

```
def get_tokenized_sentence_and_indices(iterator):
        tokenized_sentence = next(iterator)
        token_indices = [vocab[token] for token in tokenized_sentence]
        return tokenized_sentence, token_indices

tokenized_sentence, token_indices = get_tokenized_sentence_and_indices(my_iterator)
next(my_iterator)

print("Tokenized Sentence:", tokenized_sentence)
print("Token Indices:", token_indices)
```

```
OUTPUT
Tokenized Sentence: ['introduction', 'to', 'nlp']
Token Indices: [14, 19, 1]
Tokenized Sentence: ['basics', 'of', 'pytorch']
Token Indices: [11, 15, 2]
Tokenized Sentence: ['named', 'entity', 'recognition', 'with', 'pytorch']
Token Indices: [6, 4, 16, 9, 2]
```

Special Tokens & Build vocab from iterator

```
tokenizer_en = get_tokenizer('spacy', language='en_core_web_sm')
tokens = []
max_length = 0
```

```
for line in lines:
    tokenized_line = tokenizer_en(line)
    tokenized_line = ['<bos>'] + tokenized_line + ['<eos>']
    tokens.append(tokenized_line)
    max_length = max(max_length, len(tokenized_line))
```

["IBM taught me tokenization"]

['<bos>', 'IBM', 'taught', 'me', 'tokenization', '<eos>']

["Special tokenizers are ready and they will blow your mind"]

['<bos>', 'Special', 'tokenizers', 'are', 'ready', 'and', 'they', 'will', 'blow', 'your', 'mind', '<eos>']