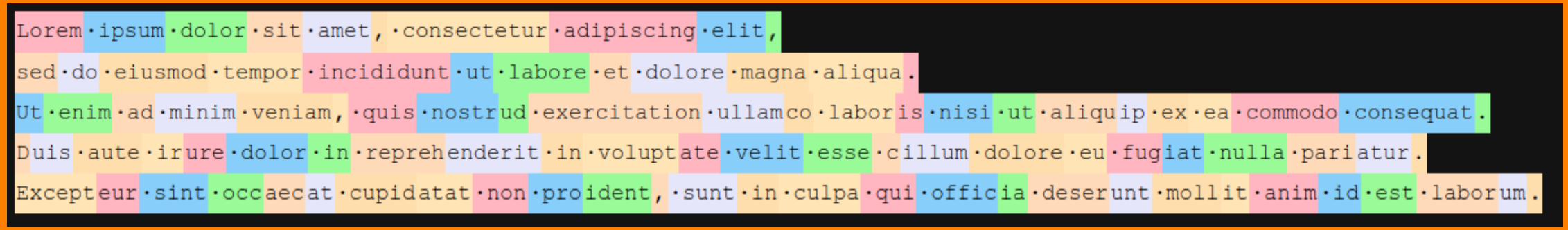
Understanding Tokenization in LLMs

Author: Mariusz Wołoszyn

What is BPE Tokenization?

Byte Pair Encoding (BPE) is a subword tokenization technique commonly used in natural language processing (NLP), particularly in large language models (LLMs). It balances vocabulary size and efficiency by breaking words into smaller subword units. This allows models to handle rare words, out-of-vocabulary words, and different language variations more effectively.



Is BPE Designed or Trained from Data?

BPE tokenizers are trained from data. They learn subword units based on frequency patterns in a given dataset. This makes them adaptable but also dataset-dependent—different datasets result in different tokenization rules.

> Having said that, it's common to add specialy crafted tokens that are designed to serve particular purpose.

How Does BPE Work?

BPE is a data-driven algorithm that builds a tokenizer based on the training data. It follows these steps:

1. **Start with individual characters**: Initially, every character in the dataset is treated as a separate token.
2. **Count character pair frequencies**: The algorithm scans the dataset and finds the most frequently occurring adjacent character pair.
3. **Merge the most frequent pair**: The identified pair is merged into a single token.
4. **Repeat the process**: Steps 2-3 are repeated until a predefined vocabulary size is reached.

For example, given the word "lower" in the dataset, if "lo" and "we" are frequent, BPE might tokenize it as `["lo", "wer"]` instead of `["l", "o", "w", "e", "r"]`.

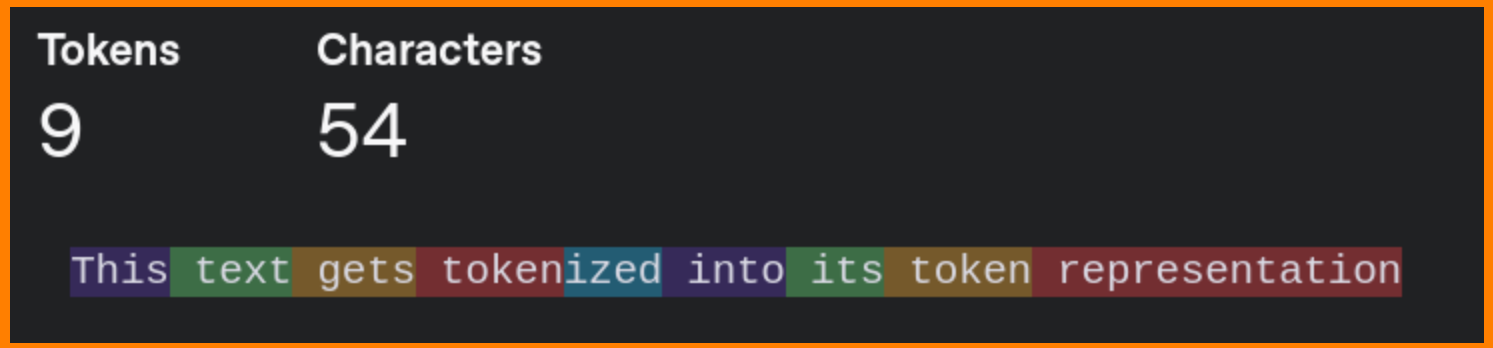
Why Do Different LLMs Have Different Tokenizers If They Use BPE?

Even though many LLMs use BPE, their tokenizers differ due to:

- **Dataset differences**: Models like GPT, LLaMA, and T5 are trained on different corpora, leading to different token merges.
- **Vocabulary size choices**: Some models use larger vocabularies to reduce sequence length, while others keep them smaller for efficiency.
- **Pre-tokenization strategies**: Some tokenizers apply extra processing, such as handling special characters or case normalization differently.

Why is BPE Used in Practice?

- **Efficient representation**: BPE reduces the number of tokens needed while still maintaining a relatively compact vocabulary.
- **Better handling of rare words**: Instead of treating every rare word as a unique token, BPE breaks it down into meaningful subword units.
- **Multilingual capabilities**: It enables models to work well across different languages by learning common subword patterns.



Tokenizer efficiency in English

- 1 token ~= 4 chars
- 1 token ~= ¾ words
- 100 tokens ~= 75 words

  or

- 1-2 sentence ~= 30 tokens
- 1 paragraph ~= 100 tokens
- 1,500 words ~= 2048 tokens

1 word = 1/0.75 ≈ 1.33 tokens

Romance Languages

Language	Code	Mistral	Llama	Phi	Gemma
Catalan	ca	1.844	1.764	2.051	1.639
French	fr	1.698	1.591	1.971	1.483
Italian	it	1.822	1.681	2.018	1.561
Portuguese	pt	1.827	1.726	2.100	1.560
Romanian	ro	2.132	2.038	2.542	1.792
Spanish	es	1.764	1.651	2.080	1.503

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

West-Germanic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Dutch	nl	1.924	1.854	2.285	1.703
English	en	1.424	1.429	1.502	1.283
German	de	1.953	1.791	2.382	1.650

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

North-Germanic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Danish	da	2.159	2.088	2.320	1.791
Swedish	sv	2.093	1.940	2.561	1.991

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

Baltic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Latvian	lv	2.985	2.942	3.101	2.351
Lithuanian	lt	2.973	2.897	3.281	2.339

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

East-Slavic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Russian	ru	2.556	2.321	6.313	2.185
Ukrainian	uk	2.607	2.342	6.439	2.254

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

South-Slavic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Bulgarian	bg	2.469	2.333	5.627	2.111
Croatian	hr	2.481	2.391	2.659	2.081
Serbian	sr	2.406	2.023	5.838	2.112
Slovenian	sl	2.381	2.271	2.540	1.987

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

West-Slavic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Czech	cs	2.487	2.315	3.046	2.063
Polish	pl	2.548	2.305	3.108	2.113
Slovak	sk	2.586	2.484	2.970	2.100

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

Uralic Languages

Language	Code	Mistral	Llama	Phi	Gemma
Estonian	et	2.945	2.898	3.030	2.448
Finnish	fi	3.353	3.275	3.392	2.622
Hungarian	hu	3.012	2.659	3.530	2.358

Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

Other Languages

Language	Code	Mistral	Llama	Phi	Gemma
Basque	eu	2.820	2.760	2.820	2.299
Greek	el	5.804	5.729	6.285	2.631
Irish	ga	2.473	2.438	2.431	2.090

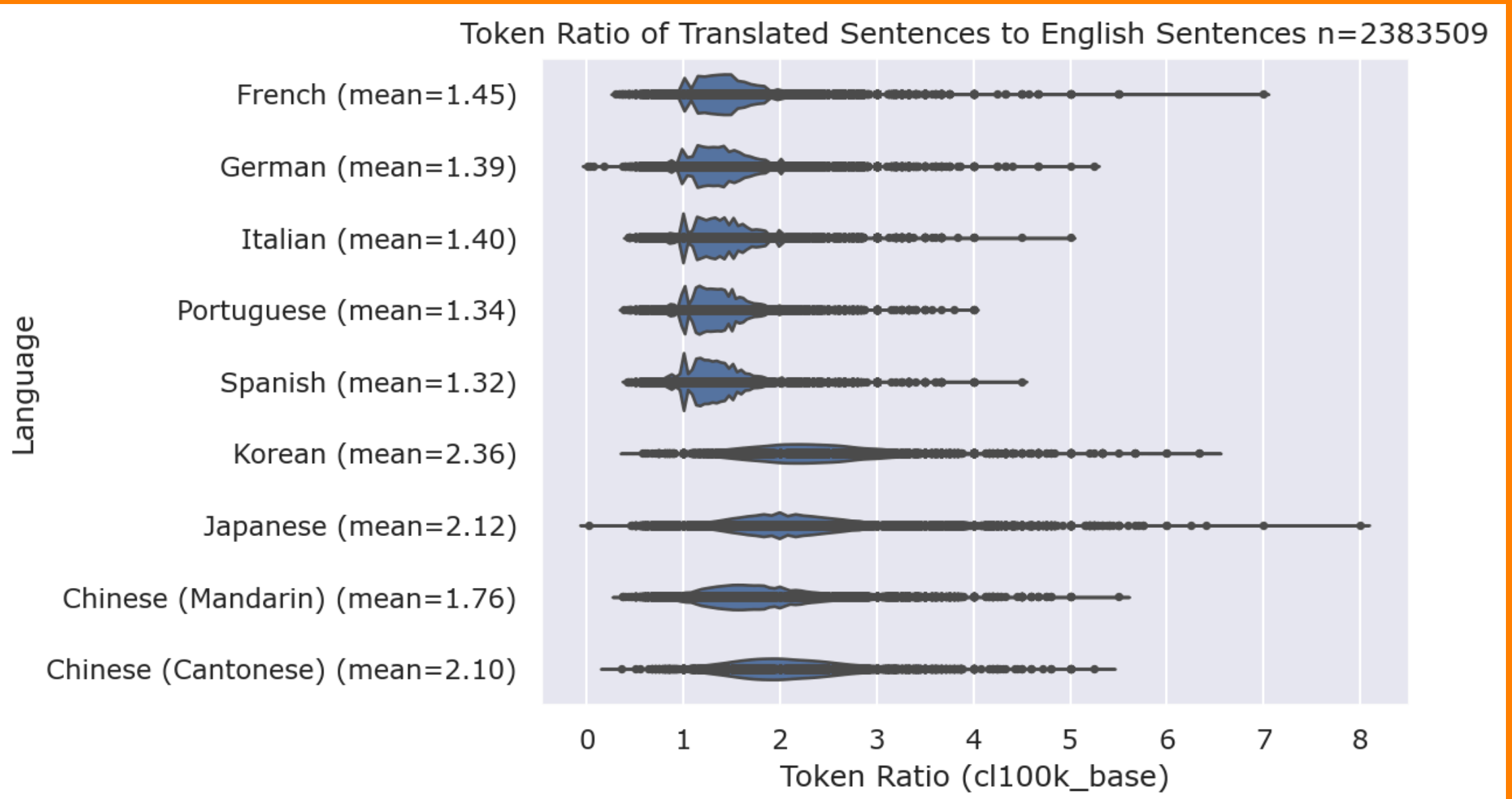
Source: https://occiglot.eu/posts/eu_tokenizer_perfomance/

Chinese, Japanese, Korean...

For a language like Japanese that has both Kanji and Kana, sentences can be up to 8x the number of tokens of the English equivalent, but average at 2.12x.

- Mandarin 1.76x
- Cantonese has 2.10x
- Korean 2.36x.

Source: https://tonybaloney.github.io/posts/cjk-chinese-japanese-korean-llm-ai-best-practices.html?utm_source=chatgpt.com


Token Ratio of Translated Sentences to English Sentences n=2383509
French (mean=1.45)
German (mean=1.39)
Italian (mean=1.40)
Portuguese (mean=1.34)
Spanish (mean=1.32)
Korean (mean=2.36)
Japanese (mean=2.12)
Chinese (Mandarin) (mean=1.76)
Chinese (Cantonese) (mean=2.10)
Language
0
1
2
3
4
5
6
7
8
Token Ratio (cl100k_base)

Tiktoken

```
import tiktoken
encoding = tiktoken.encoding_for_model("gpt-4o-mini")
encoding.encode("tiktoken is great!")
```

Count tokens

```
def num_tokens_from_string(string: str, encoding_name: str) -> int:
    """Returns the number of tokens in a text string."""
    encoding = tiktoken.get_encoding(encoding_name)
    num_tokens = len(encoding.encode(string))
    return num_tokens
```

Source: https://cookbook.openai.com/examples/how_to_count_tokens_with_tiktoken

Tokenizer libraries

- Python: tiktoken
- .NET / C#: SharpToken, TiktokenSharp
- Java: jtokkit
- Golang: tiktoken-go
- Rust: tiktoken-rs

OpenAI tokenizer tool: https://platform.openai.com/tokenizer

How to Initialize a Tokenizer in Transformers Library

Hugging Face's `transformers` library provides pre-trained tokenizers for different models. Here’s how you can load and use them:

For GPT-2:

```
from transformers import GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
text = "Byte Pair Encoding is useful."
tokens = tokenizer.tokenize(text)
print(tokens)
```

For BERT:

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
tokens = tokenizer.tokenize("Tokenization is important.")
print(tokens)
```

For LLaMA:

```
from transformers import LlamaTokenizer

tokenizer = LlamaTokenizer.from_pretrained("meta-llama/Llama-2-7b")
tokens = tokenizer.tokenize("How do LLMs work?")
print(tokens)
```

For T5:

```
from transformers import T5Tokenizer

tokenizer = T5Tokenizer.from_pretrained("t5-small")
tokens = tokenizer.tokenize("Sequence-to-sequence models use tokenization.")
print(tokens)
```

Conclusion

BPE is a widely used tokenization method that efficiently balances vocabulary size and text representation. Despite being a common approach, different models have unique tokenizers due to dataset variations, vocabulary sizes, and processing techniques. Understanding BPE helps in effectively working with LLMs and optimizing text processing workflows.