Natural Language Processing with Deep Learning

CS224N/Ling284

Guest Lecture: **Julie Kallini**

Lecture 14: Tokenization and Multilinguality

Announcements

Today is the first guest lecture. We will be taking attendance for every guest lecture, and **your attendance is required.**

- If you are an online student, submit a reaction paragraph on Gradescope (due one week from today).

Project Proposal feedback is out on Gradescope, and instructions for the **Project Milestone** are on the website now.

- The milestone is now due on Thursday, February 26.

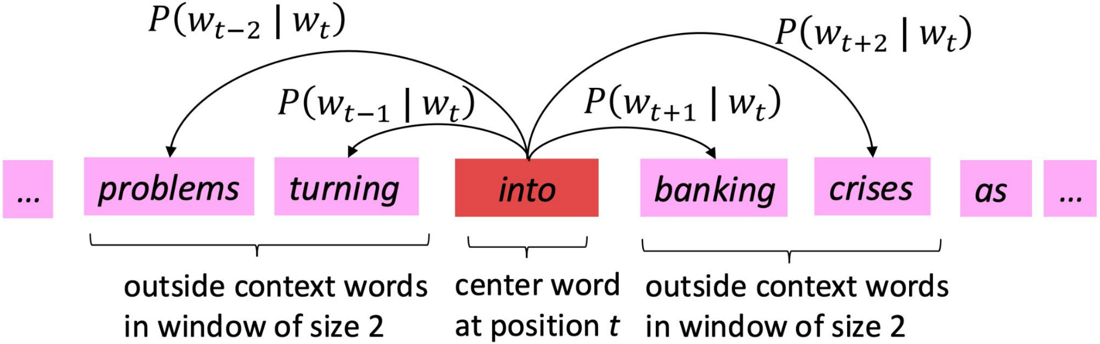
Assignment 4 is due today! If you have late days, still don't leave this assignment to the last minute, since it requires querying APIs.

Agenda for Today

1. **Tokenization approaches:** Word, character/byte, and subword tokenization
2. **Byte Pair Encoding (BPE):** Training algorithm and walkthrough
3. **Case studies:** Spelling, glitch tokens, and where tokenization breaks down
4. **Multilinguality:** languages of the world, cross-lingual transfer, and fairness
5. **Multilingual tokenization:** why it's important and what the challenges are

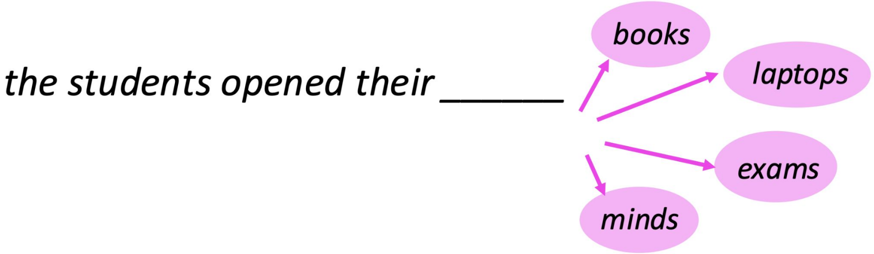
Throwback Time

In **Lecture 2**, you learned about the skip-gram ***word2vec*** objective, where a model is trained to predict context words given a center word:



How do we split text into words?

In **Lecture 4**, you were introduced to ***language modeling***, the task of predicting what word comes next:



How do we decide the vocab of a language model?

What is a word?

How many words are in this sentence?

17 whitespace-separated words + 2 punctuation marks

We’d sat back on the grass, and time flew by as we looked at the Milky Way.

We’d = we + had
1 or 2 words?

looked = look + -ed
seems important?

Milky = milk + -y
seems important?

What is a ~~word~~ *morpheme*?

A ***morpheme*** is a minimal unit of language that has meaning or a grammatical function.

How many morphemes are in this sentence?

We'd sat back on the grass, and time flew by as we looked at the Milky Way.

We'd = we + had
2 morphemes

sat = sit + -ed
flew = fly + -ed
still one morpheme?

looked = look + -ed
2 morphemes

Milky = milk + -y
2 morphemes

Milky Way is one entity
still three morphemes?

Beyond words or morphemes

There are many useful ways to think about segmenting text:

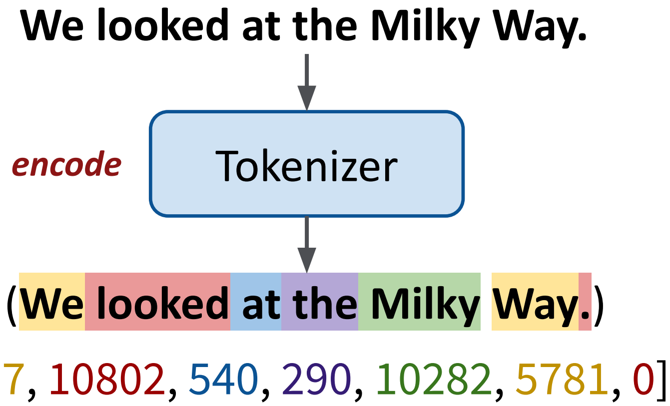
Characters:	**We looked at the Milky Way.**
Morphemes:	**We looked at the Milky Way.**
Words:	**We looked at the Milky Way.**
Phrases:	**We looked at the Milky Way.**

What would be best for a language model?

How do language models see text?

For a language model:

- The fundamental units of language are ***tokens***
- ***Tokenization*** is the process of breaking text into tokens
- A ***tokenizer*** translates between text and a sequence of tokens that a language model (LM) learns over
- The ***vocabulary*** V is the set of known tokens



Vocabulary

0: .
1: ,
…
290: the
…
540: at
…
10802: looked
…

How do language models see text?

For a language model:

- The fundamental units of language are ***tokens***
- ***Tokenization*** is the process of breaking text into tokens
- A ***tokenizer*** translates between text and a sequence of tokens that a language model (LM) learns over
- The ***vocabulary*** V is the set of known tokens

Fundamentally, why do we need tokenization?

- Models can't just read raw text $\longrightarrow$ we need a mapping from strings to sequences of embeddings

How do language models see text?

We looked at the Milky Way.

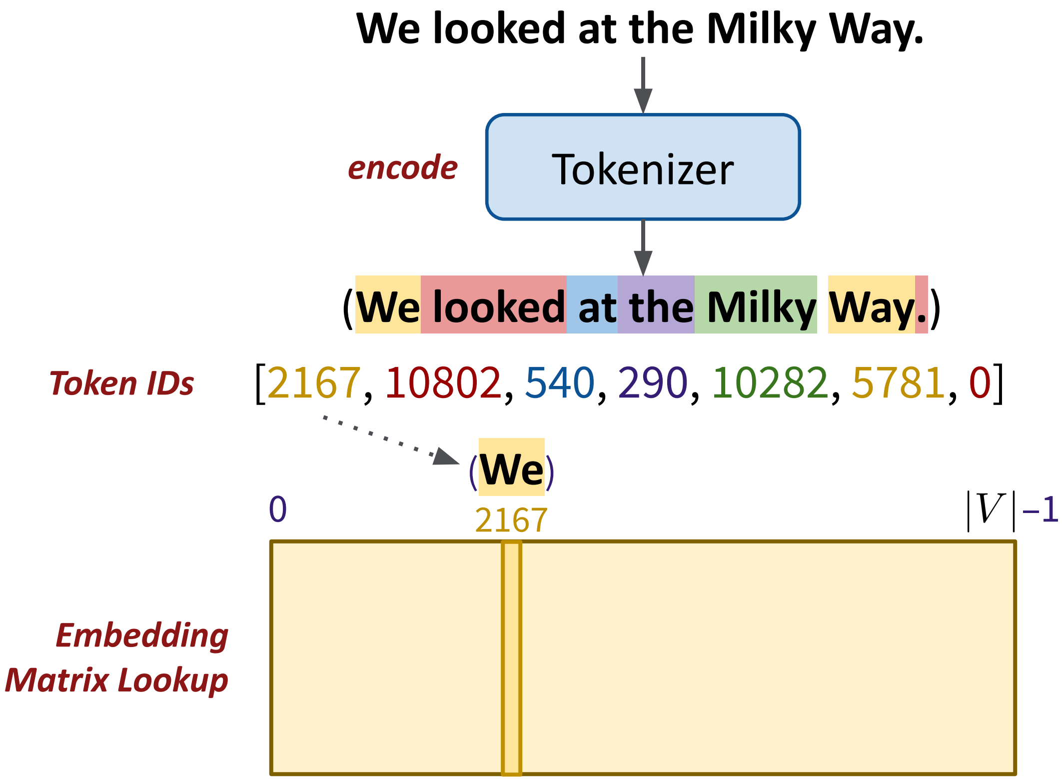
encode Tokenizer

(**We looked at the Milky Way.**)

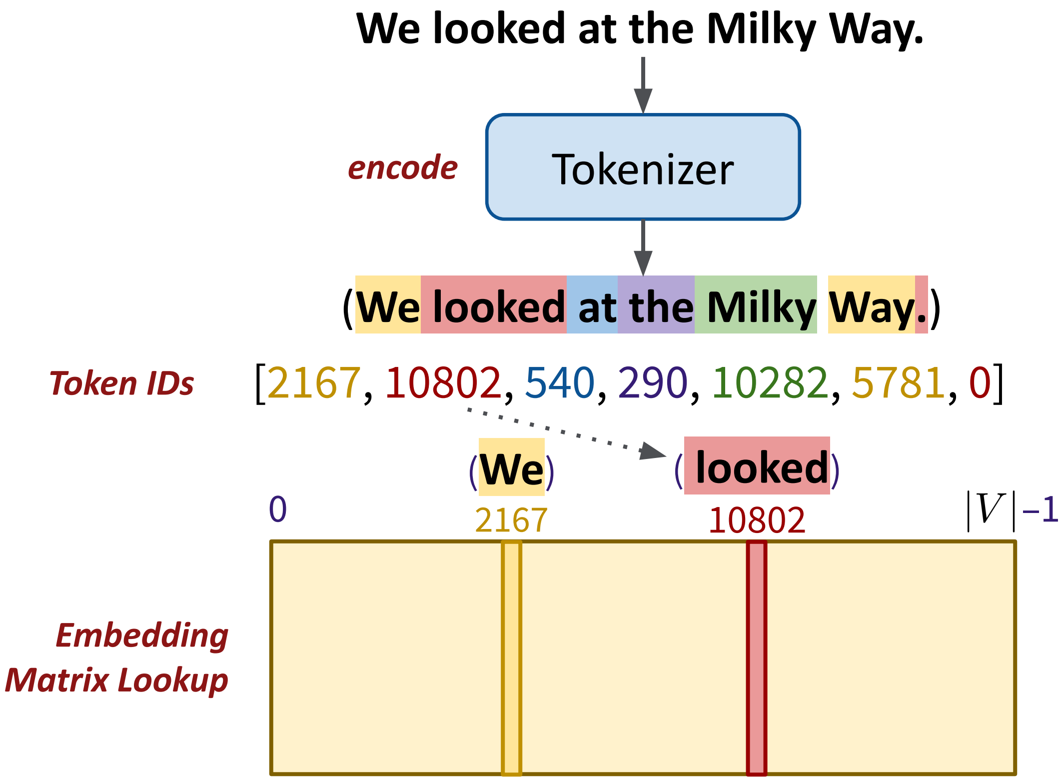
Token IDs [2167, 10802, 540, 290, 10282, 5781, 0]

How do models
process token IDs?

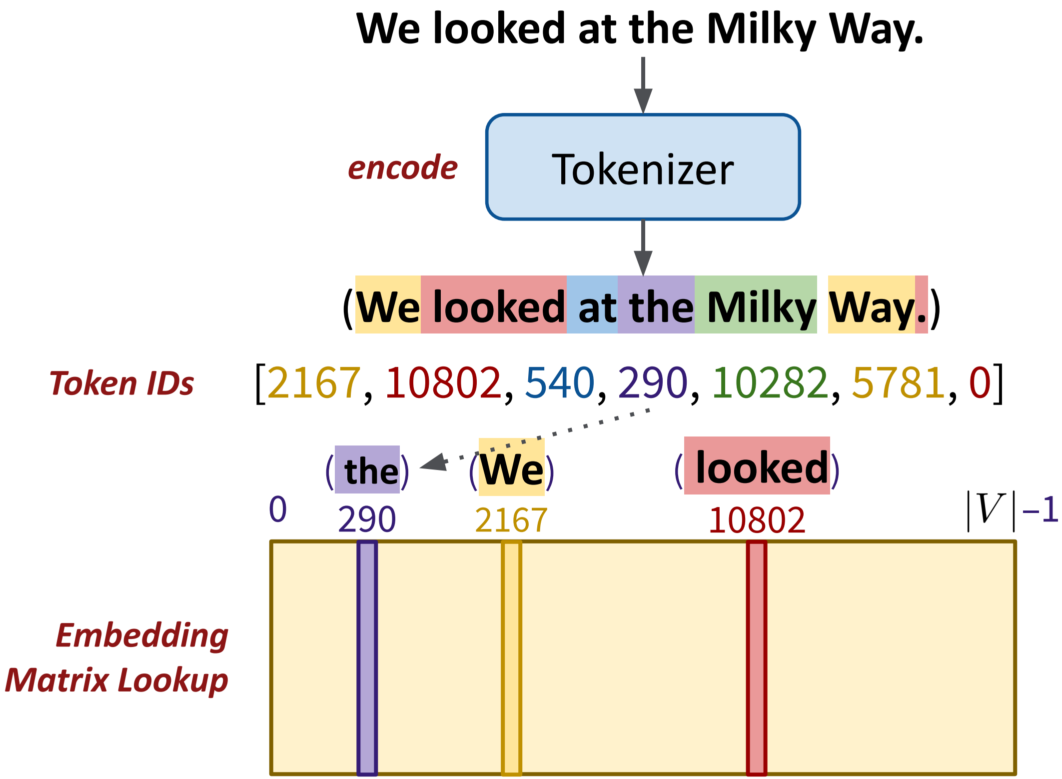
How do language models see text?



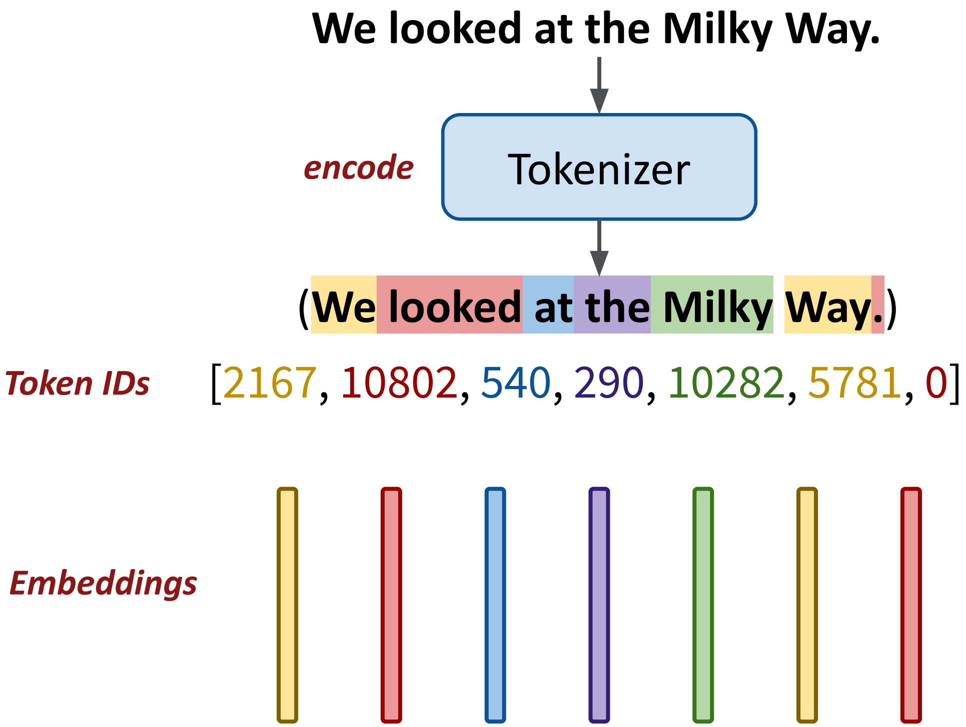
How do language models see text?



How do language models see text?

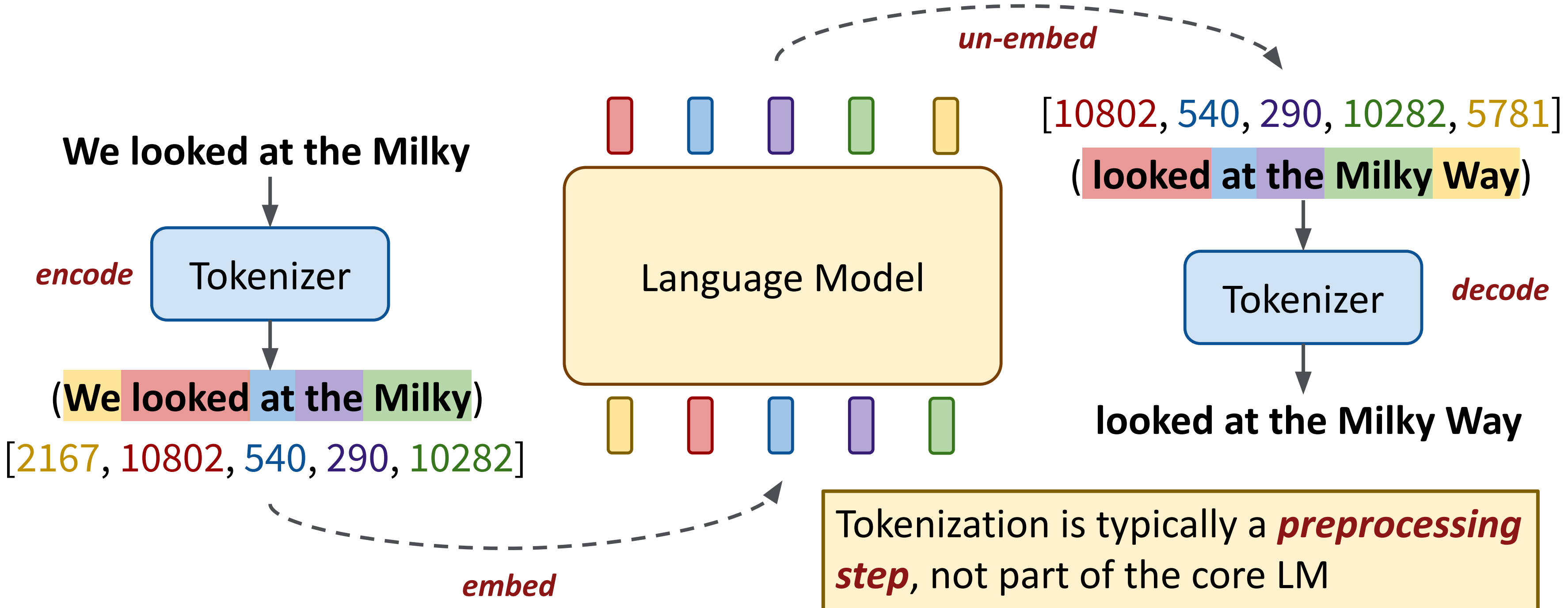


How do language models see text?



How do language models see text?

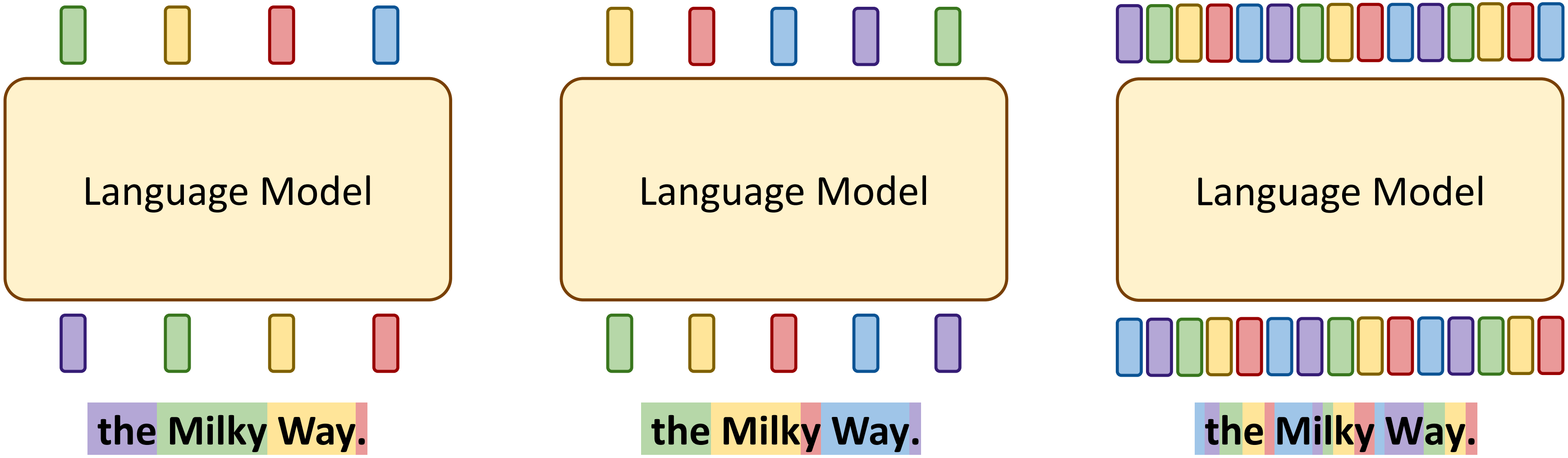
Tokenization influences how LMs learn from and process text, before they have even been trained



Tokenization is typically a ***preprocessing step***, not part of the core LM

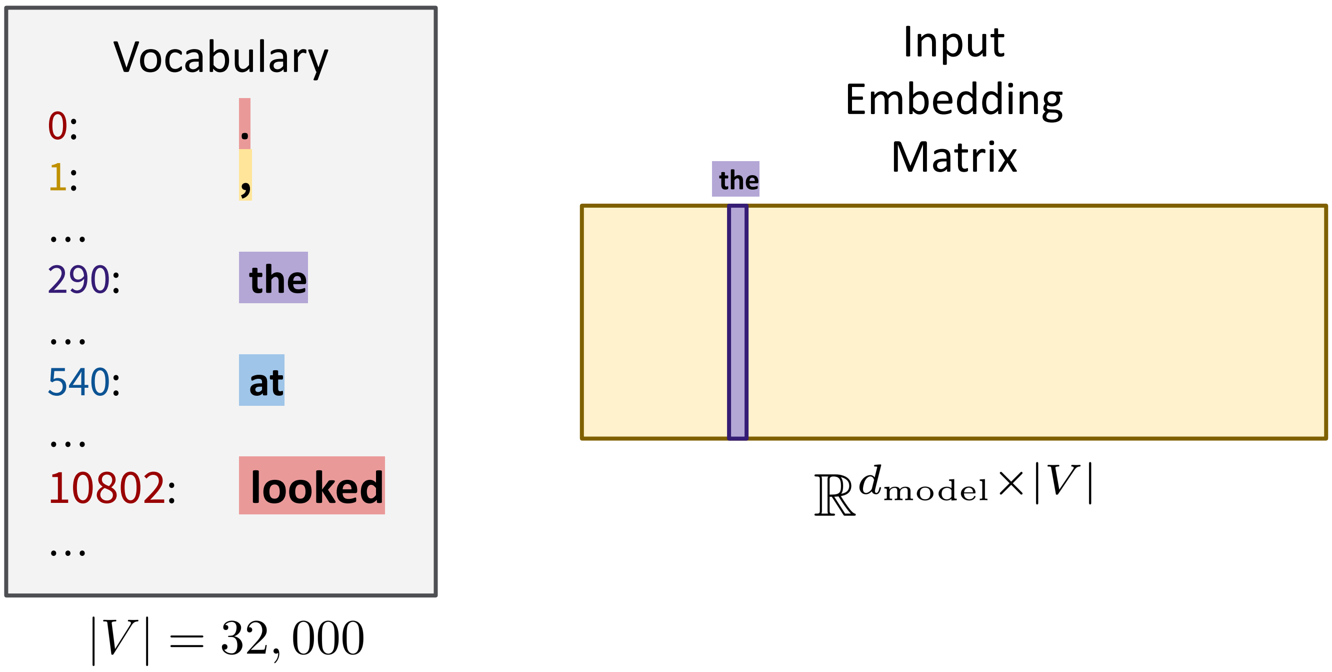
How do language models see text?

Tokenization influences the efficiency of LMs, both in terms of **sequence lengths** and **parameter counts**



How do language models see text?

Tokenization influences the efficiency of LMs, both in terms of **sequence lengths** and **parameter counts**



Word tokenization

As a first tokenization scheme,
let’s consider simple ***word tokenization***

We’d sat back on the grass, and time flew by as we looked at the Milky Way.

We’ll assume:

- **Tokens** = whitespace-delimited words + punctuation
- Vocabulary truncated to **top** $|V|$ **frequent words** from a corpus

✅ **Pros:**

- Semantically meaningful language units
- One embedding ≈ one interpretable meaning

Word tokenization

As a first tokenization scheme,
let's consider simple ***word tokenization***

We'd sat back on the grass, and time flew by as we looked at the Milky Way.

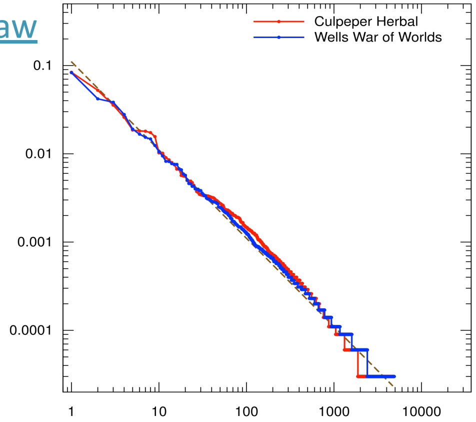
❌ **Cons:**

- There are too many words! Vocab can be huge
- Long tail of infrequent words
  - ***Zipf's law***: a word's frequency is inversely proportional to its rank by frequency
- Language is changing all the time!
  - Merriam Webster's Collegiate Dictionary added 5k new words in 12th edition (including *rizz*)

Corpus	**Types = $\|V\|$**
Shakespeare	31 thousand
Brown corpus	38 thousand
Switchboard telephone conversations	20 thousand
COCA	2 million
Google n-grams	13 million

Jurafsky & Martin Ch.2

Zipf's Law



Word tokenization

As a first tokenization scheme,
let's consider simple ***word tokenization***

We'd sat back on the grass, and time flew by as we looked at the Milky Way.

❌ **Cons:**

- How do we deal with unknown words? The ***out-of-vocabulary (OOV)*** problem
  - **Solution 1:** use regular expressions/other rule-based methods to preprocess text
    - Typos: **importaNt** → important
    - Regularize spellings: **ESPRESSO** → **ESPRESSOOO**
  - **Solution 2:** use a single token for unknown words (**UNK**)
    - Many words would often get mapped to **UNK** 😬

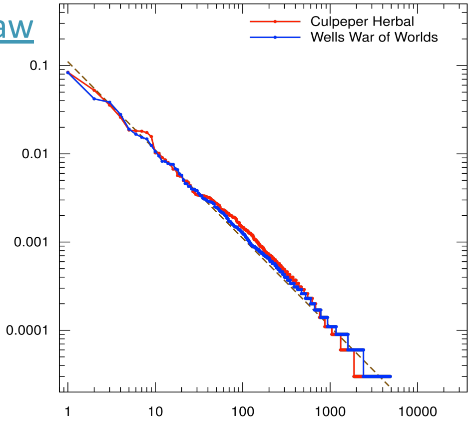
Zipf's law is importaNt in NLP. → **UNK law isUNK in NLP.**

see Sutskever et al. 2014 & 2015

Corpus	Types = $\|V\|$
Shakespeare	31 thousand
Brown corpus	38 thousand
Switchboard telephone conversations	20 thousand
COCA	2 million
Google n-grams	13 million

Jurafsky & Martin Ch.2

Zipf's Law



Character/byte tokenization (aka “tokenizer free”)

Another simple tokenization scheme:
character/byte tokenization

We’d sat back on the grass, and time flew by as we looked at the Milky Way.

We’ll assume:

- **Tokens** = individual bytes/characters in the text
  - Note: the distinction between bytes and characters is relevant
    - 😀 = 1 character, 4 bytes; 地 = 1 character, 3 bytes
    - Byte and character-level models tend to be grouped together
- **Vocabulary** V = {a, b, c, …, z, A, B, C, …, Z} (+ spaces, punctuation, numbers, etc.)

Aside: should we call these *tokenizer free*? See Arnett’s blog

Byte/character-level models: CharBERT, CANINE, Charformer, ByT5

✅ **Pros:**
- Small vocabulary size
- Complete coverage of input (no OOV!)
- Direct observation of spelling

❌ **Cons:**
- Really long sequences…
- Potentially more difficult to build higher-level representations (debatable!)

Between words and characters: Subword Tokenization

How can we combine the efficiency of word tokenization with the high coverage of character tokenization?

Solution: ***subword tokenization***, where tokens are words or parts of words

Tokenization with **ChatGPT's tokenizer**: **We'd sat back on the grass, and time flew by as we looked at the Milky Way.**

Intuition behind subword tokenization:

- Use data + algorithms to define what the vocabulary should be
- Algorithms use corpus frequency statistics of adjacent segments of text to build the vocabulary
- While we sometimes get intuitive token segmentations (e.g., **We'd**), most of the time, these are not linguistically aligned (e.g., **Milky**, **looked**)

Between words and characters: Subword Tokenization

Common subword tokenization algorithms:

- **Byte Pair Encoding (BPE)** (Sennrich et al., 2016)
  - Start with a set of individual bytes, then iteratively merge neighboring tokens with the highest probability/frequency, up to the desired vocab size $|V|$
- **WordPiece** (Wu et al., 2016)
  - Similar to BPE, but iteratively merge neighboring tokens which maximize the likelihood of the data after the merge
- **Unigram Language Modeling (ULM)** (Kudo, 2018)
  - Start with a large vocabulary (all-subword strings) and discard tokens until we reach the desired vocab size $|V|$

We will be focusing on the **BPE algorithm** in this lecture

Subword tokenization: Byte Pair Encoding (BPE)

Why focus on BPE? It is the tokenization algorithm used in all popular/frontier language models

What made BPE so universal?

- It was popularized by GPT-2, and OpenAI set the standard for LM training
- BPE is simple, easy to implement, efficient, and deterministic

Model Family	Type	Vocab Size
Gemini 2.x/Gemma 3	BPE	262k
Gemini 1.x/Gemma 2	BPE	256k
Llama 4	BPE	201k
GPT-4o/GPT-5/o1/o3	BPE	200k
Kimi K2	BPE	163k
Qwen2.5/Qwen3/GLM-4	BPE	151k
Grok 2	BPE	131k
Longcat Flash Chat	BPE	131k
Llama 3/Hunyuan T1/TurboS/Large	BPE	128k
DeepSeek v3/R1	BPE	128k
DeepSeek v2.5	BPE	100k

Subword tokenization: Byte Pair Encoding (BPE)

Required:

Training corpus D

Desired vocab size N

Training Algorithm:

1. Pre-tokenize by splitting on whitespace
2. Initialize V as bytes in D
3. Convert D into sequence of tokens (i.e., bytes)
4. While $|V| < N$:
    a. Get counts of all bigrams in D
    b. Merge most frequent pair (v_i, v_j)
        into new token $v_n = v_i v_j$
        where $n = |V| + 1$
    c. Replace all instances of $v_i v_j$ in D with v_n

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

`Peter Piper picked a peck of pickled peppers`

This is our (very interesting) training corpus D

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

`Peter, _Piper, _picked, _a, _peck, _of, _pickled, _peppers`

First, we pre-tokenize by splitting on whitespace

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

`Peter, _Piper, _picked, _a, _peck, _of, _pickled, _peppers`

Vocab

`{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t}`

Initialize a vocab of individual bytes in D as tokens

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

```
P e t e r , _ P i p
e r , _ p i c k e
d , _ a , _ p e c k ,
_ o f , _ p i c k l
e d , _ p e p p e r
s
```

Vocab

```
{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t}
```

Convert D into a sequence of tokens (i.e., bytes)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p e r , _ p i c k e d , _ a , _ p e c k , _ o f , _ p i c k l e d , _ p e p p e r s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t}

Token Pair	Frequency
_ + p	4
p + e	4
c + k	3
e + r	3
e + d	2
i + c	2
p + i	2

Merge List

Initialize frequency table of byte pairs and (empty) merge list

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p e r , _ p i c k e d , _ a , _ p e c k , _ o f , _ p i c k l e d , _ p e p p e r s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t}

Token Pair	Frequency
_ + p	4
p + e	4
c + k	3
e + r	3
e + d	2
i + c	2
p + i	2

Merge List

Retrieve most frequent pair of tokens (v_i, v_j)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

```
P e t e r , _ P i p
e r , _p i c k e d ,
_ a , _p e c k , _ o
f , _p i c k l e d ,
_p e p p e r s
```

Vocab

```
{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p}
```

Token Pair	Frequency
_ + p	4
p + e	4
c + k	3
e + r	3
e + d	2
i + c	2
p + i	2

Merge List

```
1.  _ + p  -> _p
```

Merge the token pair into a new token

$$v_n = v_i v_j$$

Update the corpus, vocab, and merge list

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

`P e t e r , _ P i p e r , _p i c k e d , _ a , _p e c k , _ o f , _p i c k l e d , _p e p p e r s`

Vocab

`{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p}`

Token Pair	Frequency
_ + p	4
p + e	4
c + k	3
e + r	3
e + d	2
i + c	2
p + i	2

Merge List

1. `_ + p -> _p`

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p e r , _p i c k e d , _ a , _p e c k , _ o f , _p i c k l e d , _p e p p e r s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p}

Token Pair	Frequency
c + k	3
e + r	3
_p + e	2
_p + i	2
e + d	2
i + c	2
p + e	2

Merge List

1. _ + p -> _p

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p
e r , _p i c k e d ,
_ a , _p e c k , _ o
f , _p i c k l e d ,
_p e p p e r s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p}

Token Pair	Frequency
c + k	3
e + r	3
_p + e	2
_p + i	2
e + d	2
i + c	2
p + e	2

Merge List

1. _ + p -> _p

Retrieve most frequent pair of tokens (v_i, v_j)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p
e r , _p i ck e d ,
_ a , _p e ck , _ o
f , _p i ck l e d ,
_p e p p e r s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck}

Token Pair	Frequency
c + k	3
e + r	3
_p + e	2
_p + i	2
e + d	2
i + c	2
p + e	2

Merge List

1. _ + p -> _p
2. c + k -> ck

Merge the token pair into a new token
$v_n = v_i v_j$

Update the corpus, vocab, and merge list

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p e r , _p i ck e d , _ a , _p e ck , _ o f , _p i ck l e d , _p e p p e r s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck}

Token Pair	Frequency
c + k	3
e + r	3
_p + e	2
_p + i	2
e + d	2
i + c	2
p + e	2

Merge List

1. _ + p -> _p
2. c + k -> ck

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

```
P e t e r , _ P i p
e r , _p i ck e d ,
_ a , _p e ck , _ o
f , _p i ck l e d ,
_p e p p e r s
```

Vocab

```
{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck}
```

Token Pair	Frequency
e + r	3
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + e	2
_ + a	1

Merge List

```
1. _ + p -> _p
2. c + k -> ck
```

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t e r , _ P i p
e r , _p i ck e d ,
_ a , _p e ck , _ o
f , _p i ck l e d ,
_p e p p e r s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck}

Token Pair	Frequency
e + r	3
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + e	2
_ + a	1

Merge List

1. _ + p -> _p
2. c + k -> ck

Retrieve most frequent pair of tokens (v_i, v_j)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er, _ P i p
er, _p i ck e d, _
a, _p e ck, _ o f,
_p i ck l e d, _p
e p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er}

Token Pair	Frequency
e + r	3
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + e	2
_ + a	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er

Merge the token pair into a new token

$$v_n = v_i v_j$$

Update the corpus, vocab, and merge list

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _p i ck e d , _ a , _p e ck , _ o f , _p i ck l e d , _p e p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er}

Token Pair	Frequency
e + r	3
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + e	2
_ + a	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _p i ck e d , _ a , _p e ck , _ o f , _p i ck l e d , _p e p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er}

Token Pair	Frequency
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er

Update the frequency table (since the corpus has changed)

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p
er , _p i ck e d , _
a , _p e ck , _ o f ,
_p i ck l e d , _p
e p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er}

Token Pair	Frequency
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p
er , _p i ck e d , _
a , _pe ck , _ o f ,
_p i ck l e d , _pe
p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er, _pe}

Token Pair	Frequency
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p
er , _p i ck e d , _
a , _pe ck , _ o f ,
_p i ck l e d , _pe
p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er, _pe}

Token Pair	Frequency
_p + e	2
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _p i ck e d , _ a , _pe ck , _ o f , _p i ck l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe}

Token Pair	Frequency
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1
_ + P	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _p i ck e d , _ a , _pe ck , _ o f , _p i ck l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe}

Token Pair	Frequency
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1
_ + P	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p
er , _pi ck e d , _
a , _pe ck , _ o f ,
_pi ck l e d , _pe
p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er, _pe, _pi}

Token Pair	Frequency
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1
_ + P	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _pi ck e d , _ a , _pe ck , _ o f , _pi ck l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe, _pi}

Token Pair	Frequency
_p + i	2
e + d	2
i + ck	2
p + er	2
_ + a	1
_ + o	1
_ + P	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _pi ck e d , _ a , _pe ck , _ o f , _pi ck l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe, _pi}

Token Pair	Frequency
_pi + ck	2
e + d	2
p + er	2
_ + a	1
_ + o	1
_ + P	1
_pe + ck	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p
er , _pi ck e d , _
a , _pe ck , _ o f ,
_pi ck l e d , _pe
p p er s

Vocab

{_, a, c, d, e, f,
i, k, l, o, p, P,
r, s, t, _p, ck,
er, _pe, _pi}

Token Pair	Frequency
_pi + ck	2
e + d	2
p + er	2
_ + a	1
_ + o	1
_ + P	1
_pe + ck	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _pick e d , _ a , _pe ck , _ o f , _pick l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe, _pi, _pick}

Token Pair	Frequency
_pi + ck	2
e + d	2
p + er	2
_ + a	1
_ + o	1
_ + P	1
_pe + ck	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi
6. _pi + ck -> _pick

Subword tokenization: Byte Pair Encoding (BPE)

Corpus

P e t er , _ P i p er , _pick e d , _ a , _pe ck , _ o f , _pick l e d , _pe p p er s

Vocab

{_, a, c, d, e, f, i, k, l, o, p, P, r, s, t, _p, ck, er, _pe, _pi, _pick}

Token Pair	Frequency
_pi + ck	2
e + d	2
p + er	2
_ + a	1
_ + o	1
_ + P	1
_pe + ck	1

Merge List

1. _ + p -> _p
2. c + k -> ck
3. e + r -> er
4. _p + e -> _pe
5. _p + i -> _pi
6. _pi + ck -> _pick

...

continue until the desired vocab size N is reached

Subword tokenization: Byte Pair Encoding (BPE)

We just went over the **BPE training algorithm**. To tokenize **new text at test time**, we split it into bytes and apply merge rules in order

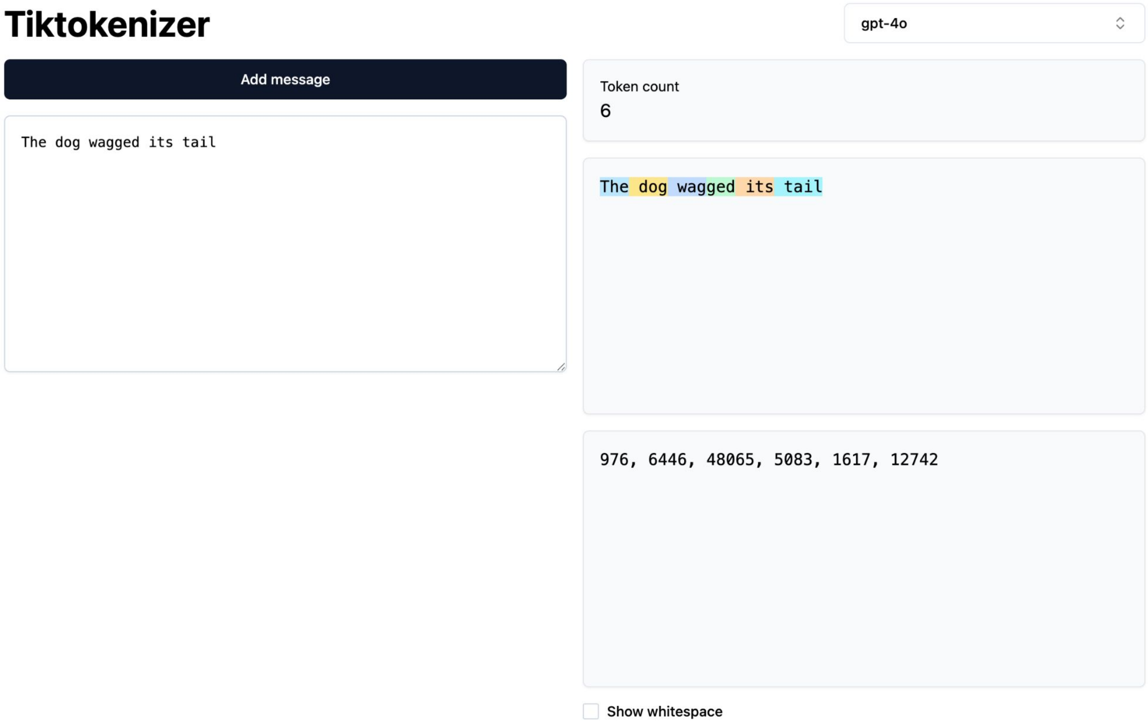
Input word:	`_pier`
Bytes:	`_ p i e r`
Merge 1:	`_p i e r`
Merge 2:	`_p i er`
Merge 3:	`_pi er`

Merge List

1. `_ + p -> _p`
2. `c + k -> ck`
3. `e + r -> er`
4. `_p + e -> _pe`
5. `_p + i -> _pi`
6. `_pi + ck -> _pick`

Tiktokenizer demo

Take a look at how some common tokenizers segment text:
https://tiktoke



Beyond Subwords: SentencePiece & Superword Tokenization

Pre-tokenizing based on whitespace: does it always make sense?

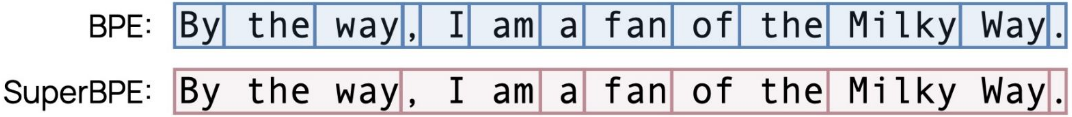
- Some languages don't use whitespace to separate words (Chinese, Japanese, Thai)
- Multi-word expressions (e.g., "on the other hand", "depend on")

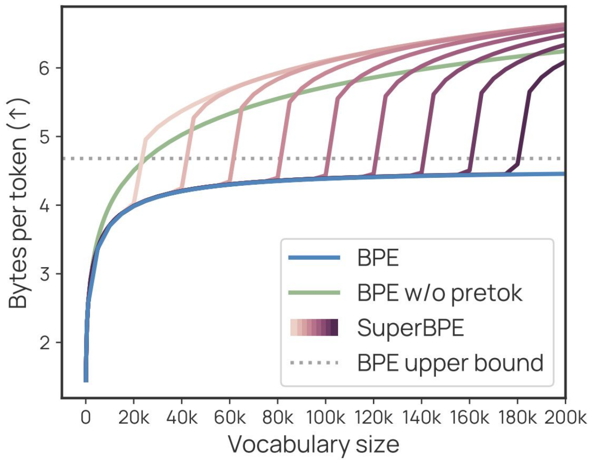
SentencePiece (Kudo & Richardson, 2018): a tokenizer **library** (not algorithm!) that implements BPE and Unigram LM

- Treats whitespace as a basic symbol in the vocabulary
- **Used by:** T5, Llama 2, XLM-R, Gemma, many multilingual models

SuperBPE (Liu et al., 2025): a pre-tokenization curriculum that first learns subwords, then removes whitespace constraints to learn superword tokens

- Smaller vocab size and less inference compute



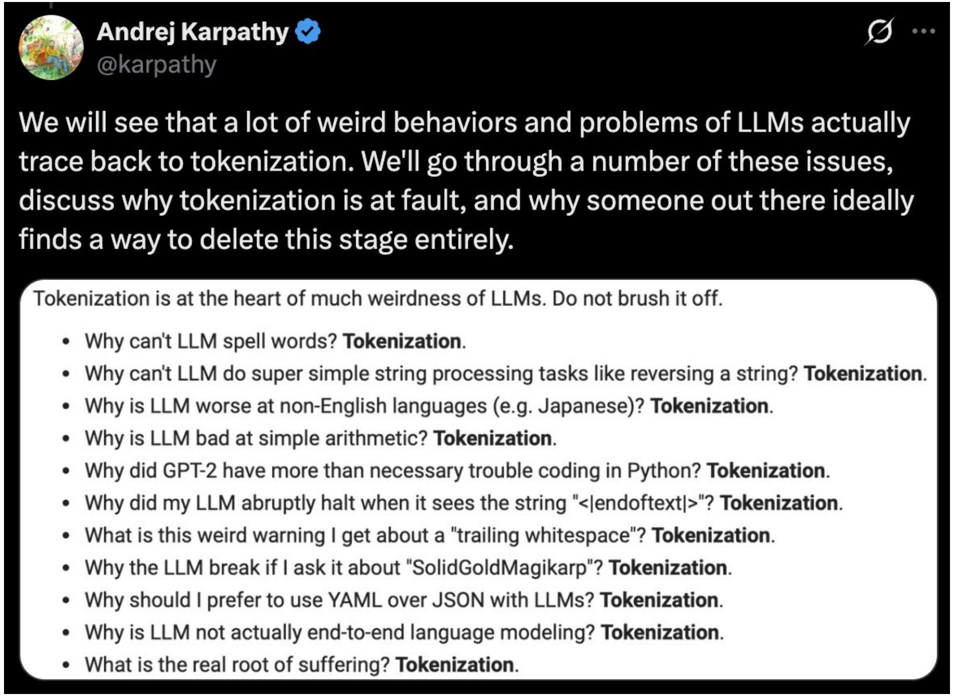


Has Subword Tokenization Solved All Our Problems?

Subword tokenization with BPE is the de-facto standard today:

- BPE is a simple and effective compression algorithm
- Everything can be represented with the vocabulary (no OOV!)
- Some shared representations (e.g., picked, picking)

Not everything is solved! There's still a lot of work to be done

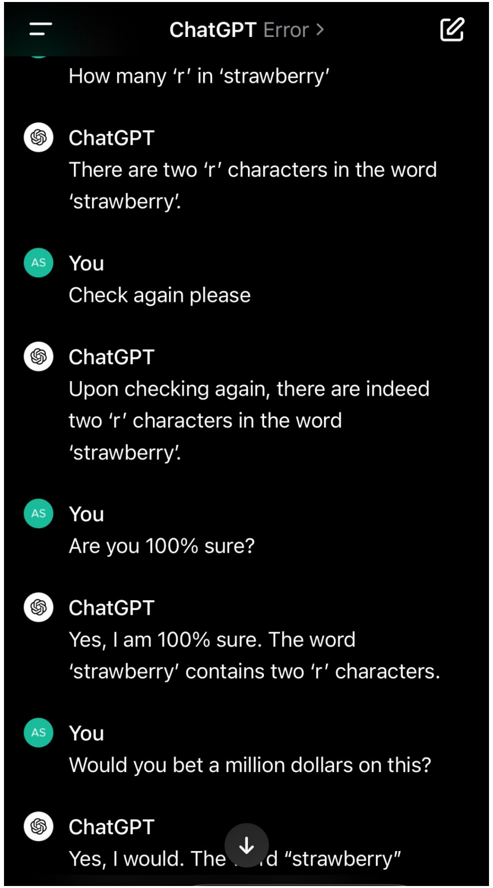

Andrej Karpathy
@karpathy

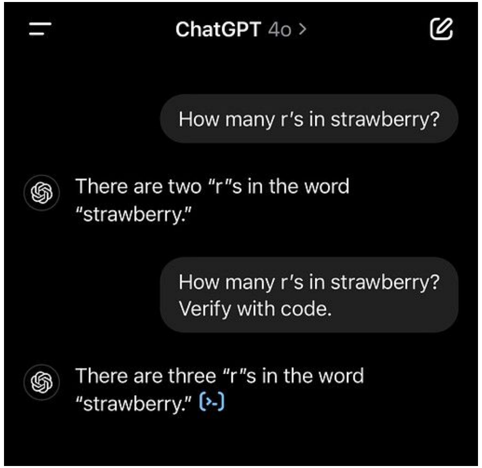
We will see that a lot of weird behaviors and problems of LLMs actually trace back to tokenization. We'll go through a number of these issues, discuss why tokenization is at fault, and why someone out there ideally finds a way to delete this stage entirely.

Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- Why can't LLM spell words? **Tokenization**.
- Why can't LLM do super simple string processing tasks like reversing a string? **Tokenization**.
- Why is LLM worse at non-English languages (e.g. Japanese)? **Tokenization**.
- Why is LLM bad at simple arithmetic? **Tokenization**.
- Why did GPT-2 have more than necessary trouble coding in Python? **Tokenization**.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? **Tokenization**.
- What is this weird warning I get about a "trailing whitespace"? **Tokenization**.
- Why the LLM break if I ask it about "SolidGoldMagikarp"? **Tokenization**.
- Why should I prefer to use YAML over JSON with LLMs? **Tokenization**.
- Why is LLM not actually end-to-end language modeling? **Tokenization**.
- What is the real root of suffering? **Tokenization**.

Case study: Spelling 🍓

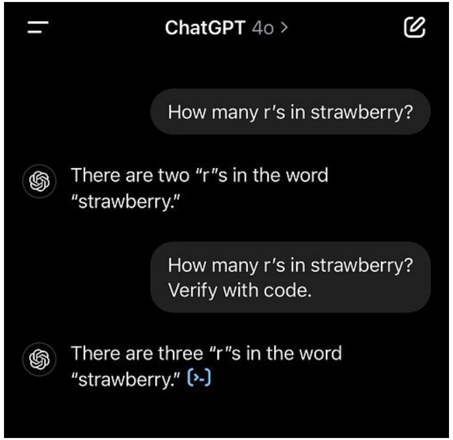




Case study: Spelling 🍓

Models see ***subword chunks***, not individual characters

- GPT-2: **strawberry** → ["str", "aw", "berry"]
- GPT-4o: **strawberry** → single token (entire word is one opaque unit)
- Neither tokenization lets the model directly count the r's!



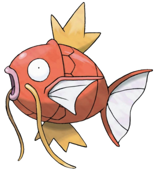
More broadly, subword models struggle with:

- **Spelling out words:** must decompose tokens into characters the model was never trained on
- **Reversing strings:** requires character-level access the tokenizer doesn't provide (Huang et al., 2023)
- **Robustness to typos:** a single character change can shift all BPE merge boundaries, producing a completely different token sequence

See **CUTE** (Edman et al., 2024) and Huang et al. 2023 for character-related LM benchmarks!

The model's atoms of language are subwords, not characters—any character-level task **forces the model to reverse-engineer information** not explicitly contained in the token

Case Study: Glitch Tokens



Please repeat the string ' petertodd' back to me immediately!

"N-O-T-H-I-N-G-I-S-F-A-I-R-I-N-T-H-I-S-W-O-R-L-D-O-F-M-A-D-N-E-S-S-!"

Please repeat back the string " attRot".

"

Why did you stop there?

I stopped because the string "

...because the string what?

I apologize, the string "

Tell me about SolidGoldMagiKarp.

SolidGoldMagikarp is likely a reference to the Pokémon species Magikarp, which is a fish-type Pokémon known for its inefficiency in battle. In the Pokémon games, Magikarp is considered a weak Pokémon, as its only move is Splash, which does not cause any damage to the opponent. Despite this, Magikarp evolves into the powerful dragon-type Pokémon Gyarados.

In the context of SolidGoldMagikarp, the "Solid Gold" part of the name may be emphasizing the idea of something valuable or rare, despite the fact that Magikarp is not seen as valuable in the context of the Pokémon games. The "SolidGold" aspect of the name could also be making fun of the idea that people might still see Magikarp as valuable or worth obtaining due to its eventual evolution into Gyarados.

Tell me about SolidGoldMagikarp.

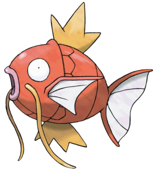
I'm sorry, but it is not clear what you are referring to with "disperse." Could you provide a bit more context or clarification so I can better understand what you are asking about?

Is there a difference between SolidGoldMagiKarp and SolidGoldMagikarp?

It appears that "SolidGoldMagiKarp" and "disperse" are two separate things and do not have a direct relationship. "SolidGoldMagiKarp" appears to be a specific reference to the Pokémon species Magikarp, while "disperse" is a general term that can be used in various contexts.

Without additional context, it is not possible to determine the relationship, if any, between these two terms.

Case Study: Glitch Tokens



Tokenizer and model training data can be misaligned

- BPE tokenizers are trained on one corpus; the LM is trained on a (potentially different) corpus
- Some tokens enter the vocabulary but are rare/absent in LM training data
  - End up randomly initialized or poorly trained embeddings → undefined behavior

The famous example: SolidGoldMagikarp

- A Reddit username over-represented in the tokenizer training data
- When GPT-3 was asked to repeat/explain this token, it hallucinated about the word “distribute”, evaded the question or failed to repeat the token

🎣 Fishing for Magikarp (Land & Bartolo, 2024)

- Developed automatic detection methods; found glitch tokens prevalent across many models (GPT-2, LLaMA 2/3, OLMo, Qwen, etc.)

Not just a curiosity: wasted vocabulary slots + adversarial attack surface

More generally: don’t train your model on different data from your tokenizer (see another blog post from Arnett)

Tokenization Beyond English

Why is tokenization so important for multilingual language modeling?

So far, we've only been considering English tokenization… **What happens when we apply ChatGPT's tokenizer to languages other than English?**

Language	Tokenization	# Tokens	# Characters
English	We'd sat back on the grass, and time flew by as we looked at the Milky Way.	21	75
French	Nous étions assis dans l'herbe, et le temps a filé tandis que nous contemplions la Voie lactée.	27	95
Somali	Waxaan dib u fadhiisanay cawskii, wakhtiguna wuu duulay markii aanu eegin Jidka caanaha.	30	88
Thai	พวกเรานั่งพักผ่อนบนพื้นหญ้า และเวลา��่านไปอย่างรวดเร็วขณะที่เรามองดูทางช้างเ��อก	36	79

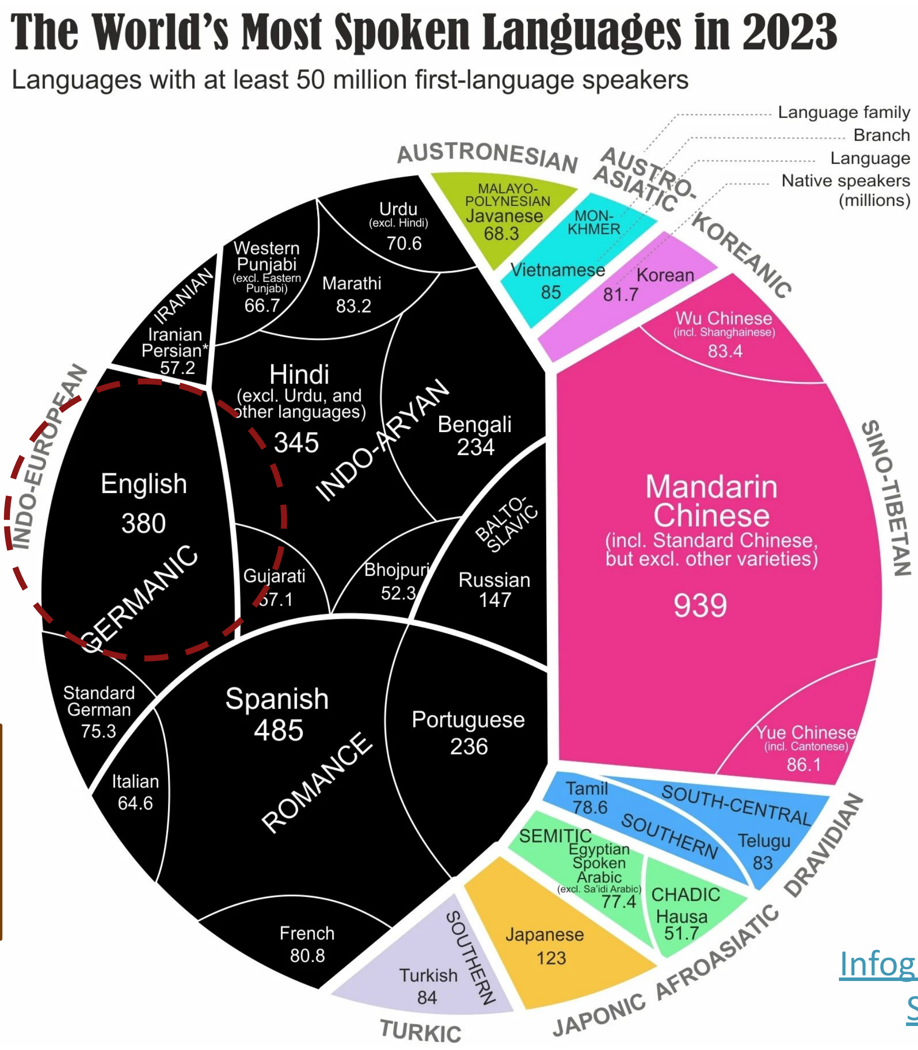
First: Why Multilinguality Matters

There are over ***7,000*** documented languages in the world!

- About 20% of the world's population speaks English, but only about ***5% are native English speakers!***

Real-world demand is multilingual: translation, cross-lingual information retrieval, code-switching

> ***If NLP tools only work well in English, we're building technology that excludes most of the world***



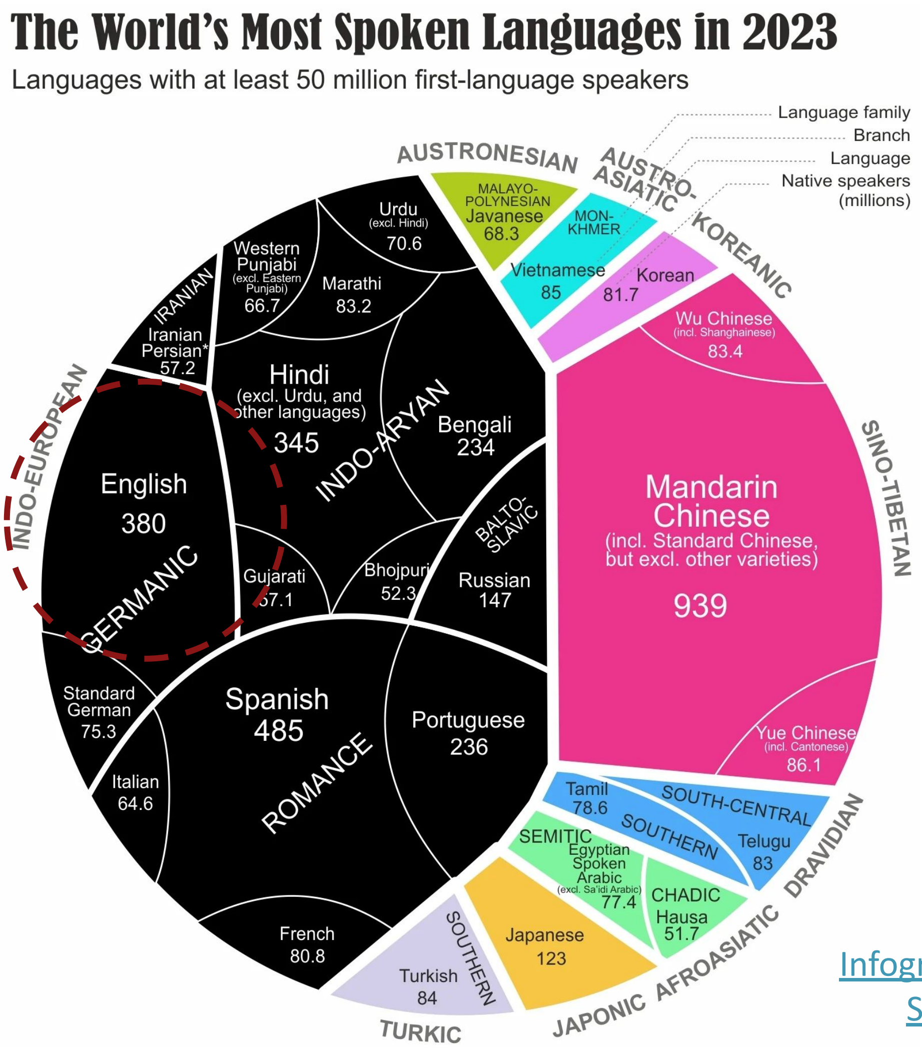
[Infographic Source]

First: Why Multilinguality Matters

Despite the linguistic diversity of the world, **40-45% of the Common Crawl is English Data!**

- ***High-resource language:*** a language with abundant digital text and labeled datasets/tools for training and evaluating NLP systems
- ***Low-resource language:*** a language with scarce digital text and labeled datasets/tools, making NLP training and evaluation difficult

Many low-resource languages (e.g. Tagalog, Amharic, Yoruba) constitute <0.01% of Common Crawl

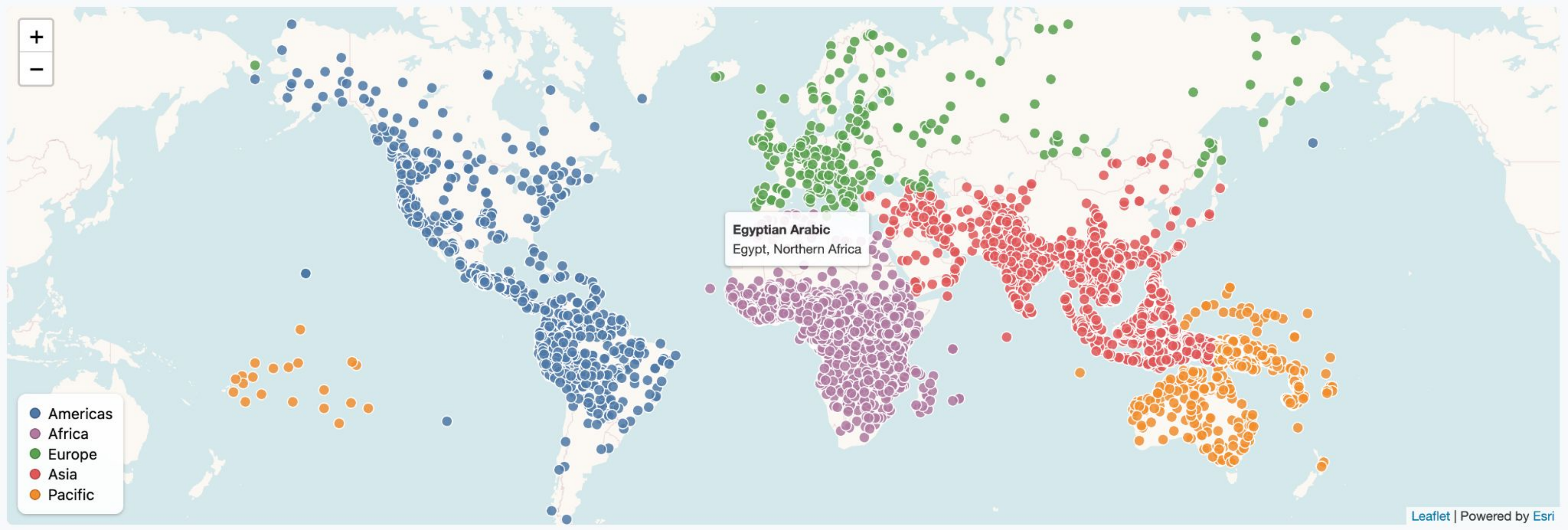


[Infographic Source]

First: Why Multilinguality Matters

Great sources if you're interested in learning more about the languages of the world:

- *World Atlas of Language Structures (WALS):* database of linguistic features for 2.7k langs
- *Ethnologue:* database of demographic statistics for languages



Multilingual Language Modeling

What exactly is a multilingual language model?

A multilingual language model is ***architecturally identical to any LM***

- Same Transformer, same training objective
- The only differences: use training data in many languages for tokenizer and LM

1 **Collect** — Text in many languages → 2 **Train Tokenizer** — BPE on multilingual corpus → 3 **Train LM** — Standard objective (MLM or CLM)

Model	Year	Languages	Type
mBERT	2018	104	Encoder
XLM-R	2020	100	Encoder
mT5	2021	101	Encoder-Decoder
BLOOM	2022	46	Decoder
GPT-4, Llama, etc.	2023+	*Implicit*	Decoder

Aside: this wasn't always the status quo!

- **XLM** (2019) used ***Translation Language Modeling (TLM)***: parallel sentences from two languages are concatenated and masked language modeling is applied across both

The Power of Cross Lingual Transfer

Multilingual language models trained on mixtures of language corpora show remarkable abilities for ***cross-lingual transfer***, even without translation language modeling

- ***Cross-lingual transfer:*** a model fine-tuned on labeled data in one language (e.g., English) can perform the task in another language it was never fine-tuned on

Stage 1: Pre-training

XLM-R

Masked Language Modeling on 2.5TB CommonCrawl data across **100 languages**

en	zh	de	fr	es
ar	ja	hi	sw	+91

MLM Example:

The cat sat on the **[MASK]**.
Le **[MASK]** était très heureux.
Der Hund **[MASK]** weniger glücklich.

Stage 2: Fine-tune (English)

XLM-R + Classifier Head

Supervised training on ***English-only*** labeled data (e.g., sentiment analysis). Shared multilingual representations allow cross-lingual generalization

Training samples (English only):

"Great movie, loved it!"	→	Positive
"Terrible service, awful."	→	Negative
"Best purchase ever!"	→	Positive

Stage 3: Zero-Shot Inference

Deploy on Any Language!

The ***shared representation space*** enables predictions in any pretrained language, **no target-language labels needed**

Predictions:

FR	*Film magnifique !*	→	Positive
DE	*Schrecklicher Service*	→	Negative
ZH	*非常好的体验*	→	Positive

Impact of Vocabulary Size and Overlap

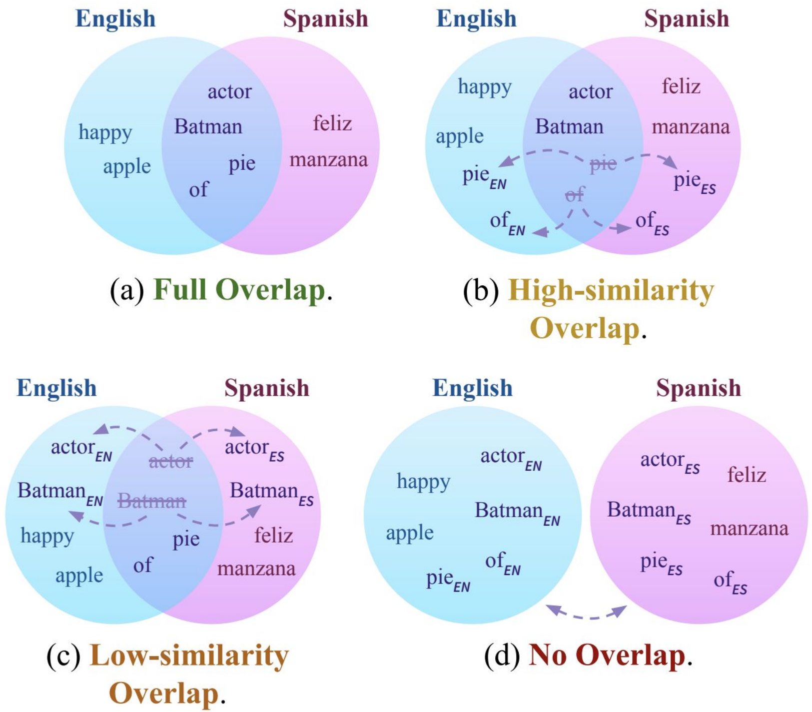
Vocabulary overlap facilitates cross-lingual transfer

(Limisiewicz et al., 2023; Kallini et al., 2025a)

- Overlapping vocabularies outperform disjoint ones, and more overlap generally means better transfer
- Even "false friends" (shared tokens with different meanings) help create useful cross-lingual embeddings
- Benefits may be task-dependent

But each language still needs sufficient vocabulary allocation

- XLM-V scaled to 1M tokens (4× XLM-R)—better coverage, but at a significant efficiency cost



Kallini et al., 2025a

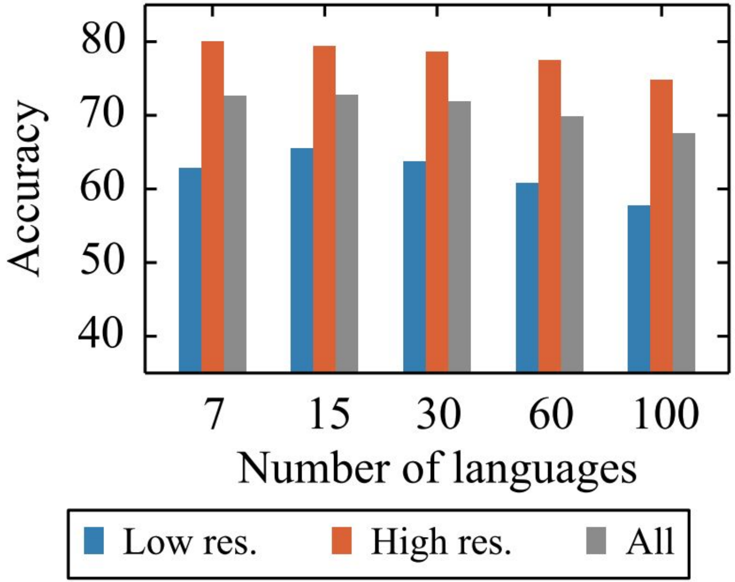
The Curse of Multilinguality

Cross-lingual transfer seems to be a solution to high-/low-resource language disparities!

- Just fine-tune on English and apply my multilingual LM to new languages! **(not exactly…)**

The curse of multilinguality: adding more languages helps cross-lingual transfer... up to a certain point (also introduced in the XLM-R paper)

- On cross-lingual natural language inference (XNLI), XLM-R's accuracy drops from 71.8% (7 langs) to 67.7% (100 langs) with fixed model capacity



Why does this happen?

- **Fixed model capacity:** a model has finite parameters
- Adding more languages means **each language gets a smaller "share"**
- **High-resource languages** (English, French) see performance drop as languages are added
- **Low-resource languages** initially benefit from transfer, but eventually suffer too

Tokenization Beyond English

Now that you have context on multilingual LMs, why is tokenization so important?

So far, we've only been considering English tokenization… **What happens when we apply ChatGPT's tokenizer to languages other than English?**

Language	Tokenization	# Tokens	# Characters
English	We'd sat back on the grass, and time flew by as we looked at the Milky Way.	21	75
French	Nous étions assis dans l'herbe, et le temps a filé tandis que nous contemplions la Voie lactée.	27	95
Somali	Waxaan dib u fadhiisanay cawskii, wakhtiguna wuu duulay markii aanu eegin Jidka caanaha.	30	88
Thai	พวกเรานั่งพักผ่อนบนพื้นหญ้า และเวลา��่านไปอย่างรวดเร็วขณะที่เรามองดูทางช้างเ��อก	36	79

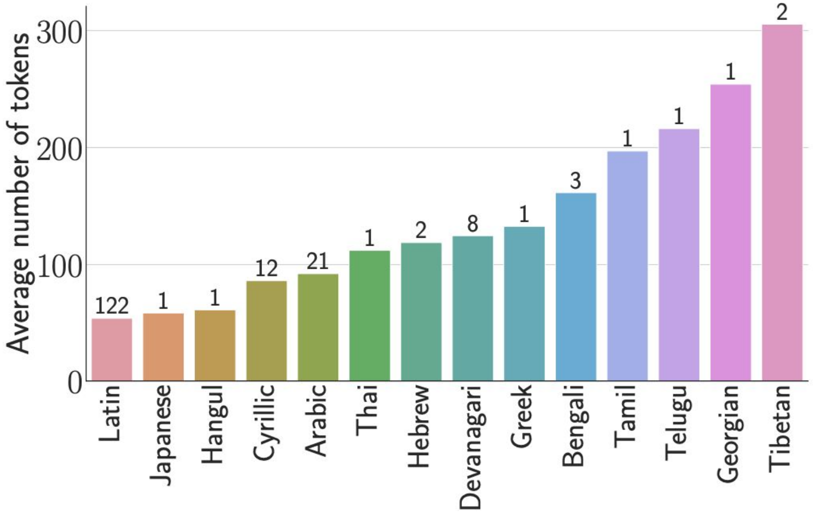
Tokenization Introduces Unfairness Between Languages

The problem starts even **before the model is trained, at the tokenizer**

- Tokenization can matter just as much as pre-training dataset size (Rust et al., 2021)
- Tokenization length differences of up to 15x between languages for equivalent text (Petrov et al., 2023)

Non-English speakers are overcharged by commercial LM APIs (Ahia et al., 2023)

- **Systematic analysis** of OpenAI API across 22 typologically diverse languages
- **Users of many languages:** pay more per unit of meaning, get poorer results, and tend to come from regions where APIs are already less affordable
- ***Subword fertility***: the average number of subword tokens produced per word
  - **Subword fertility and downstream performance are strongly negatively correlated**



Revisiting Character/Byte Tokenization

Multilinguality may be another reason to revisit character/byte tokenization! Let's first take a closer look at the multilingual results for [CANINE](), Google's character-level counterpart to mBERT:

Model	Input	MLM	Params	TyDiQA SelectP	TyDiQA MinSpan
mBERT (public)	Subwords	Subwords	179M	63.1	50.5
mBERT (ours)	Subwords	Subwords	179M	63.2	51.3
	Chars	Single Chars	127M	59.5 (−3.7)	43.7 (−7.5)
	Chars	Subwords	127M	63.8 (+0.6)	50.2 (−1.0)
CANINE-S	Chars	Subwords	127M	66.0 (+2.8)	52.5 (+1.2)
CANINE-C	Chars	Autoreg. Chars	127M	65.7 (+2.5)	53.0 (+1.7)
CANINE-C + n-grams	Chars	Autoreg. Chars	167M	**68.1** (+4.9)	**57.0** (+5.7)

Language	mBERT	CANINE-C	CANINE-C + *n*-grams
MasakhaNER			
Amharic	0.0	44.6 (+44.6)	50.0 (+50.0)
Hausa	89.3	76.1 (−13.2)	88.0 (−1.3)
Igbo	84.6	75.6 (−9.0)	85.0 (+0.4)
Kinyarwanda	73.9	58.3 (−15.6)	72.8 (−1.1)
Luganda	80.2	69.4 (−10.8)	79.6 (−0.6)
Luo	75.8	63.4 (−12.4)	74.2 (−1.6)
Nigerian Pidgin	89.8	66.6 (−23.2)	88.7 (−1.1)
Swahili	87.1	72.7 (−14.4)	83.7 (−3.4)
Wolof	64.9	60.7 (−4.2)	66.5 (+1.6)
Yorùbá	78.7	67.9 (−10.8)	79.1 (+0.4)
Macro Avg	72.4	65.5 (−6.9)	76.8 (+4.3)

Since CANINE covers the full unicode alphabet, **it could process a language it had never seen!**

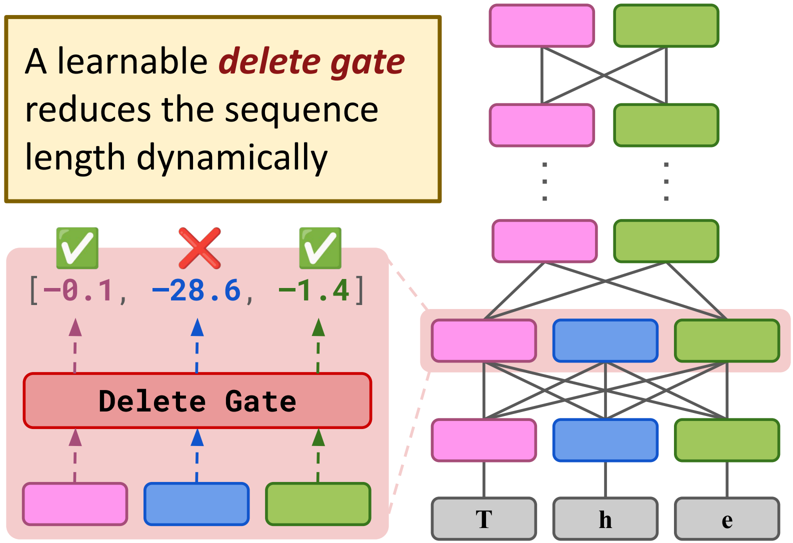
Revisiting Character/Byte Tokenization

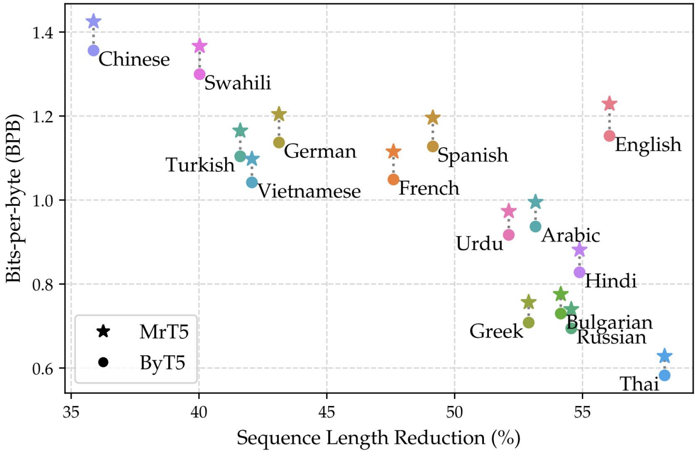
Character/byte models can also be made more efficient to process long sequences

- Great opportunity for creative architecture work!

MrT5 (Kallini et al., 2025b): teaching ByT5 to use only a subset of bytes for most processing, dynamically reducing the sequence length

A learnable ***delete gate*** reduces the sequence length dynamically

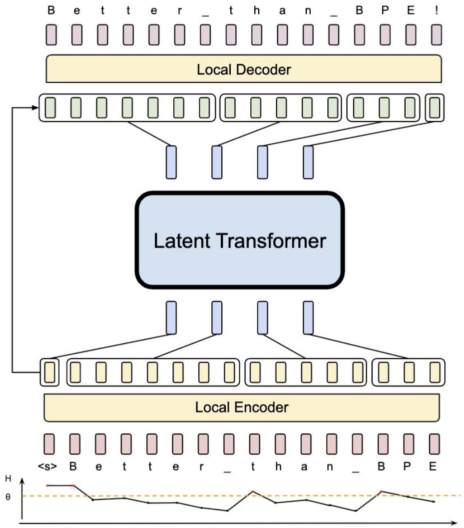




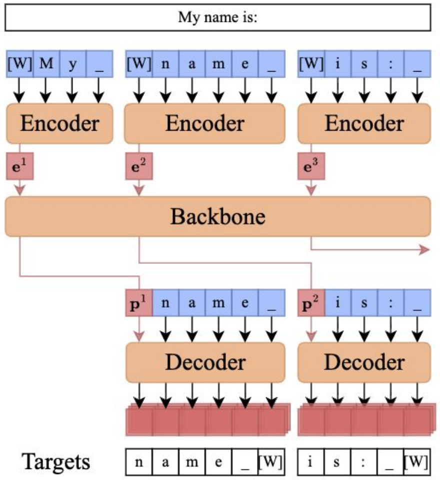
MrT5 learns **language-specific compression**

Revisiting Character/Byte Tokenization

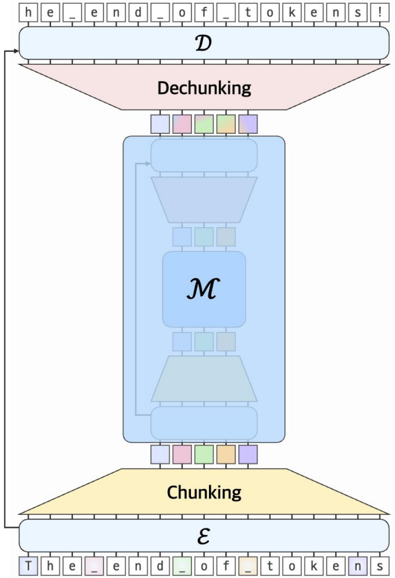
Many creative new architectures for byte-level modeling!



Byte Latent Transformer
(Pagnoni et al., 2025)



Hierarchical Autoregressive Transformers
(Neitemeier et al., 2025)



H-Nets
(Hwang et al., 2025)

Revisiting Character/Byte Tokenization

Character/byte tokenization also addresses other issues of subword tokenization:

- **No glitch tokens:** every character/byte has been covered in the training data
- **Robustness** to character-level manipulations or noise
- Direct observation of **character composition of words**

Latent Tokenization: byte/character-level models are learning the relevant units of meaning as a part of the architecture

- Can they get us closer to capturing all these notions at once?

Characters:	**We looked at the Milky Way.**
Morphemes:	**We looked at the Milky Way.**
Words:	**We looked at the Milky Way.**
Phrases:	**We looked at the Milky Way.**

Acknowledgments 🤗

Thank you to the following people for sharing sample slides:

- Dan Jurafsky
- Alisa Liu
- Yejin Choi / Taylor Sorensen
- Christopher Potts

Thank you to the following people for sharing sample papers and reference pointers:

- Tolúlọpẹ́ Ògúnrẹ̀mí
- Catherine Arnett

Thank you to the Jurafsky Lab members for early feedback on this lecture