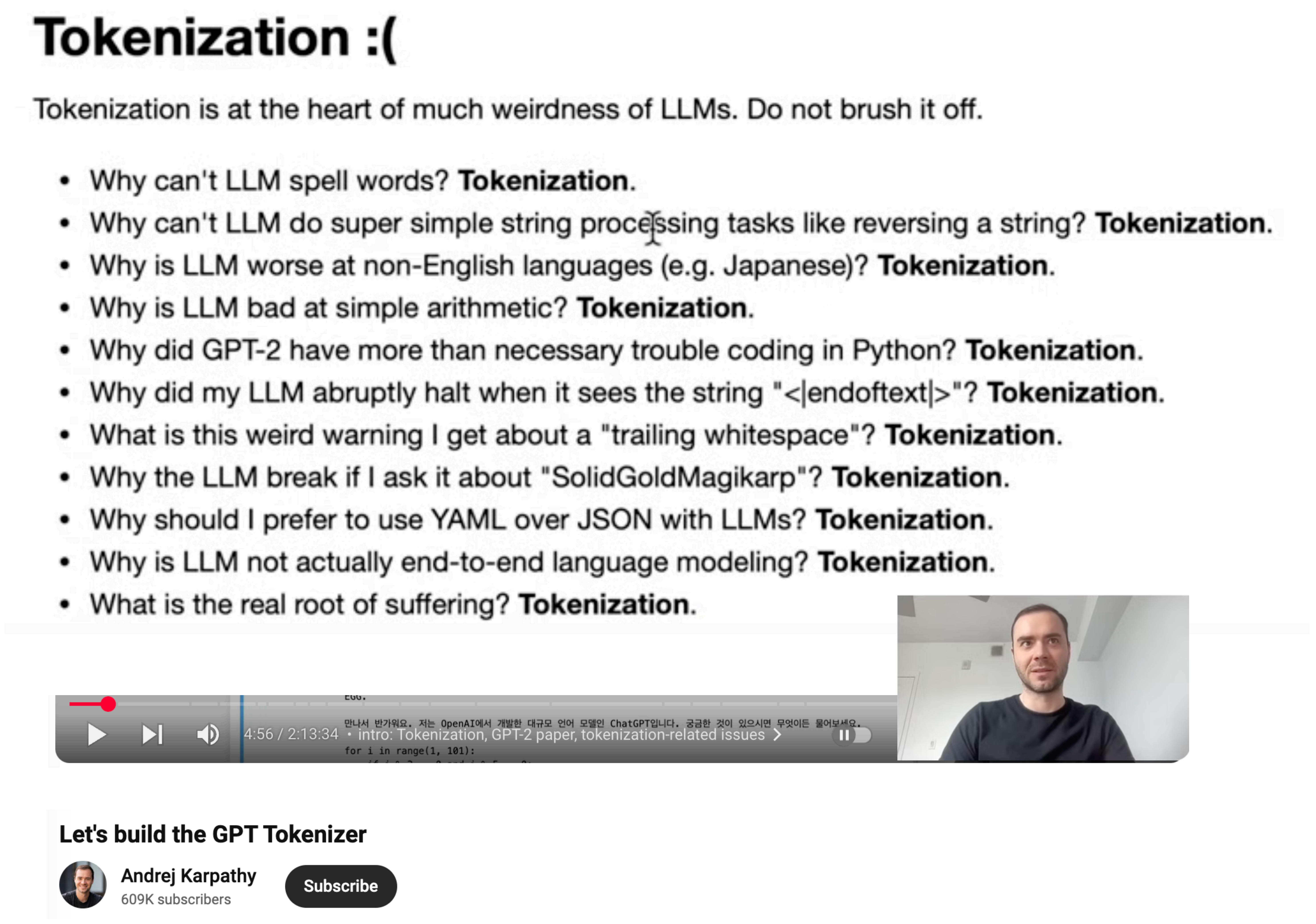
Tokenization:
How do language models see text?

Jan 27, 2025

CSE 447/517: NLP

Guest lecture from Alisa Liu

Inspiration taken from lectures of Yejin Choi, Andrej Karpathy, Sachin Kumar, Oreva Ahia



Tokenization :(

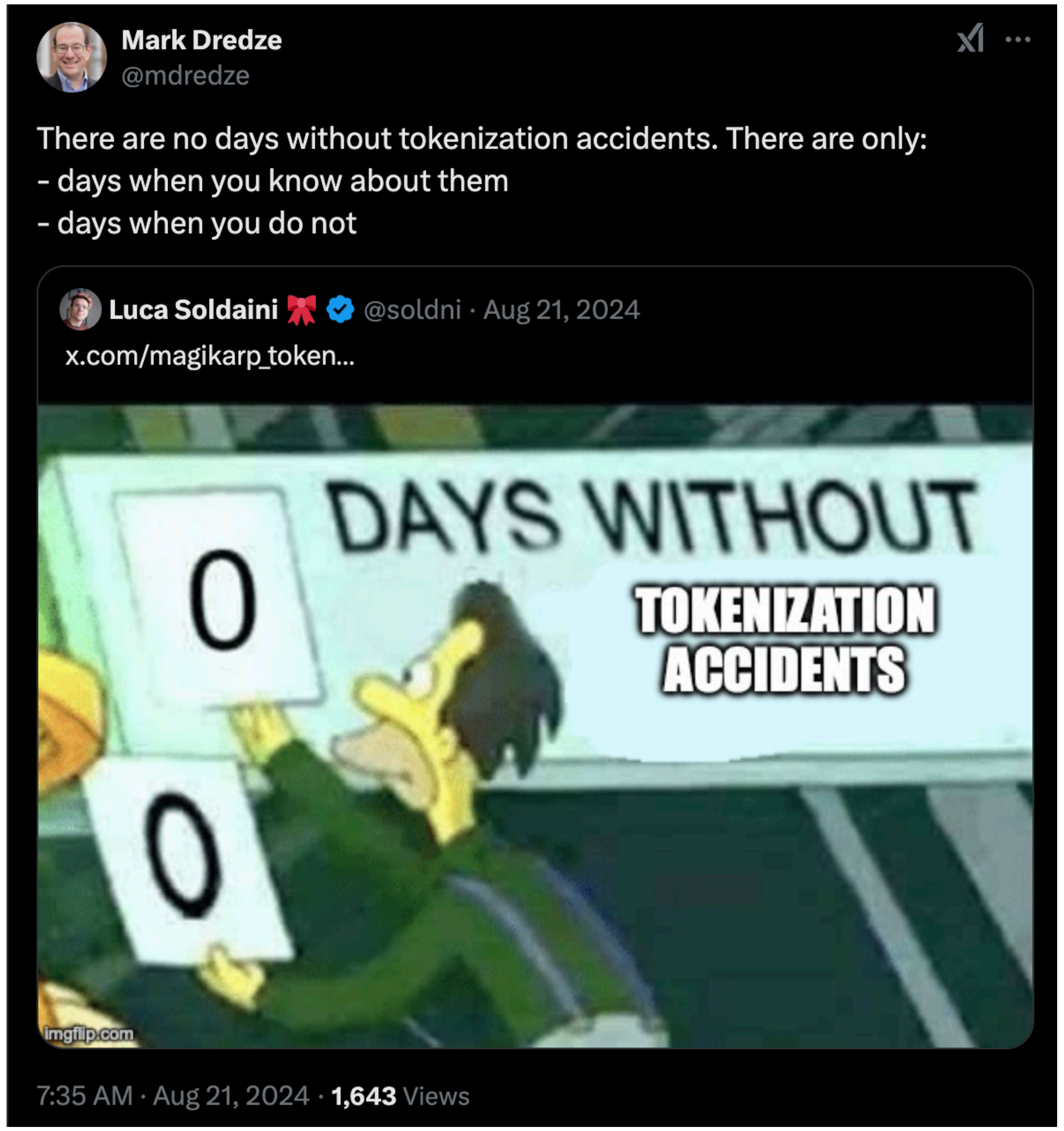
Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- 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.**

4:56 / 2:13:34 • intro: Tokenization, GPT-2 paper, tokenization-related issues
만나서 반가워요. 저는 OpenAI에서 개발한 대규모 언어 모델인 ChatGPT입니다. 궁금한 것이 있으시면 무엇이든 물어보세요.
for i in range(1, 101):

Let's build the GPT Tokenizer

Andrej Karpathy
609K subscribers



Mark Dredze
@mdredze

There are no days without tokenization accidents. There are only:
- days when you know about them
- days when you do not

Luca Soldaini 🎀 @soldni · Aug 21, 2024
x.com/magikarp_token...

7:35 AM · Aug 21, 2024 · **1,643** Views

Outline

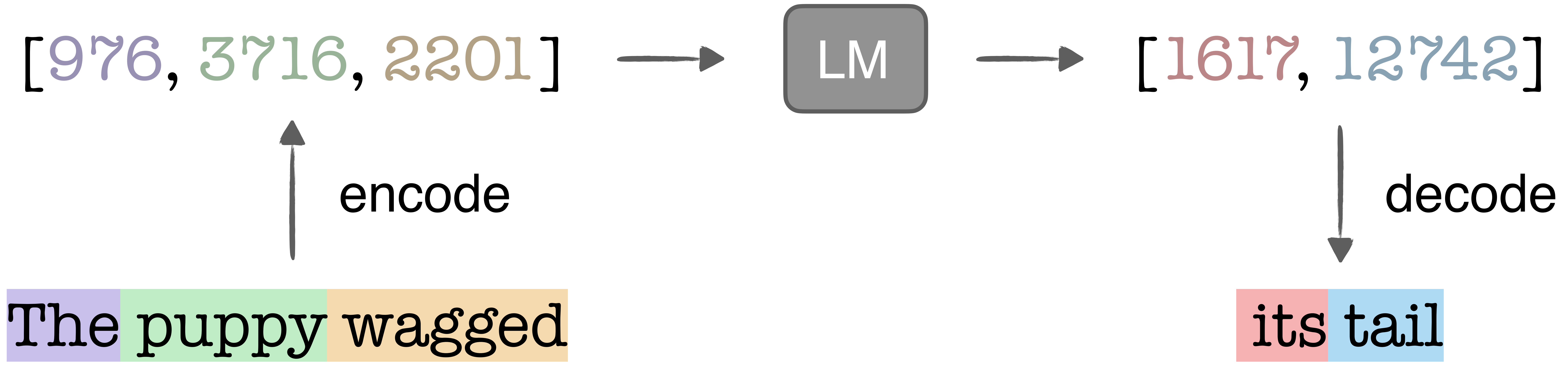
1. What is tokenization?
2. Word-level and character-level tokenizers
3. Subword-level tokenizers
4. BPE: Byte Pair Encoding
5. Variations on BPE

What is tokenization?

Token = a “word” unit with its own embedding representation

A **tokenizer** translates between text and a sequence of **tokens** that a language model (LM) learns representations over

The **vocabulary** V is the set of known tokens

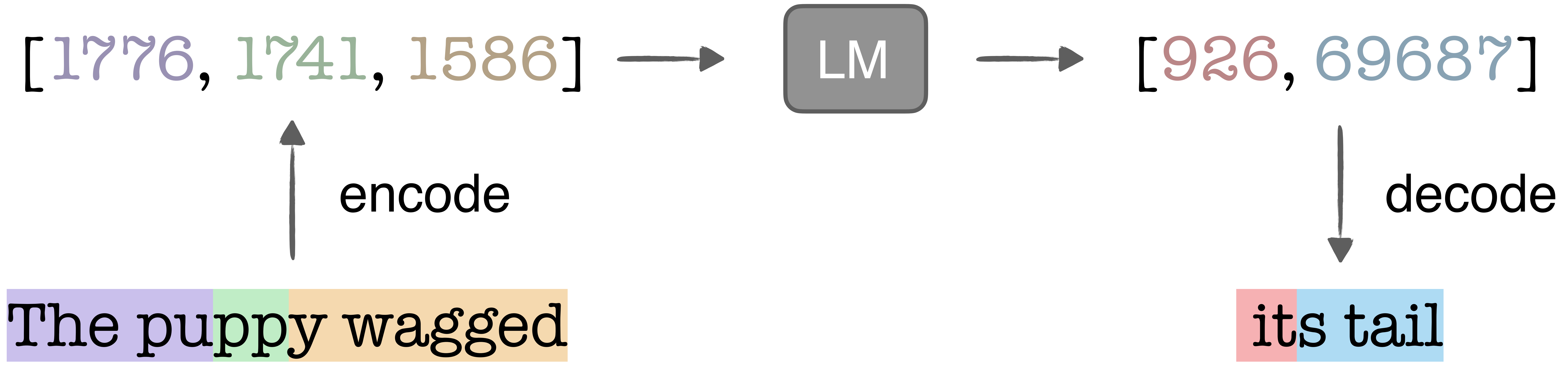


What is tokenization?

Token = a “word” unit with its own embedding representation

A **tokenizer** translates between text and a sequence of **tokens** that a language model (LM) learns representations over

The **vocabulary** V is the set of known tokens

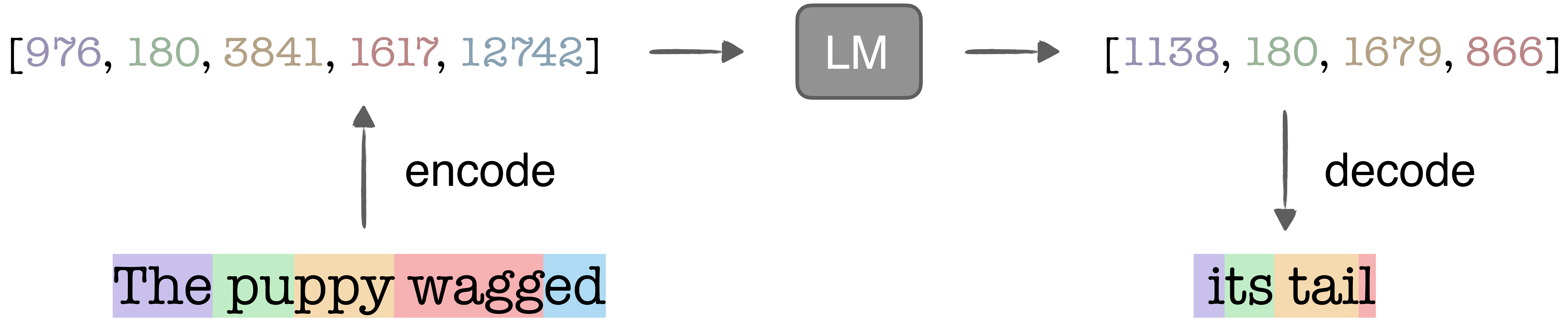


What is tokenization?

Token = a “word” unit with its own embedding representation

A **tokenizer** translates between text and a sequence of **tokens** that a language model (LM) learns representations over

The **vocabulary** V is the set of known tokens

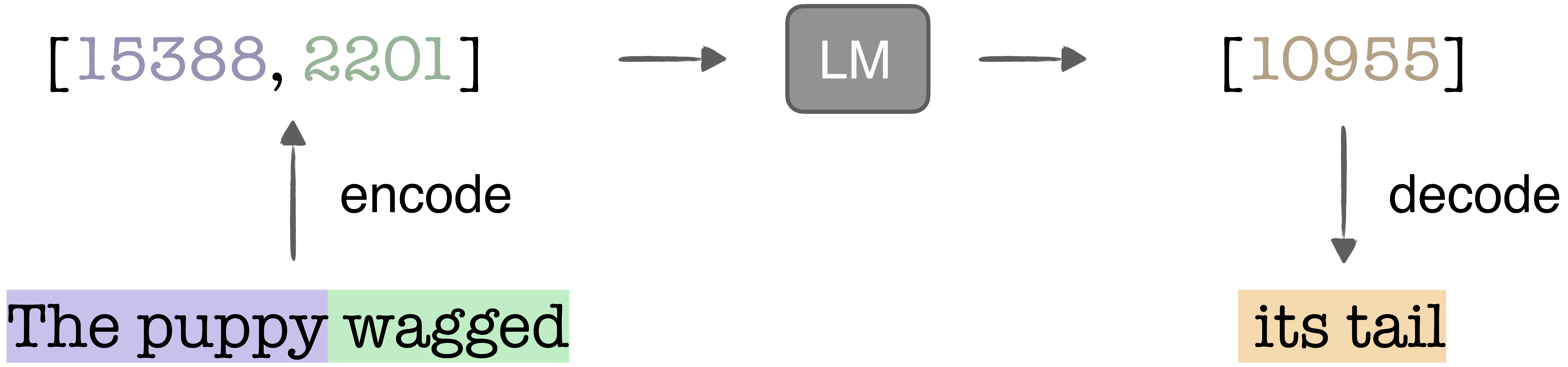


What is tokenization?

Token = a “word” unit with its own embedding representation

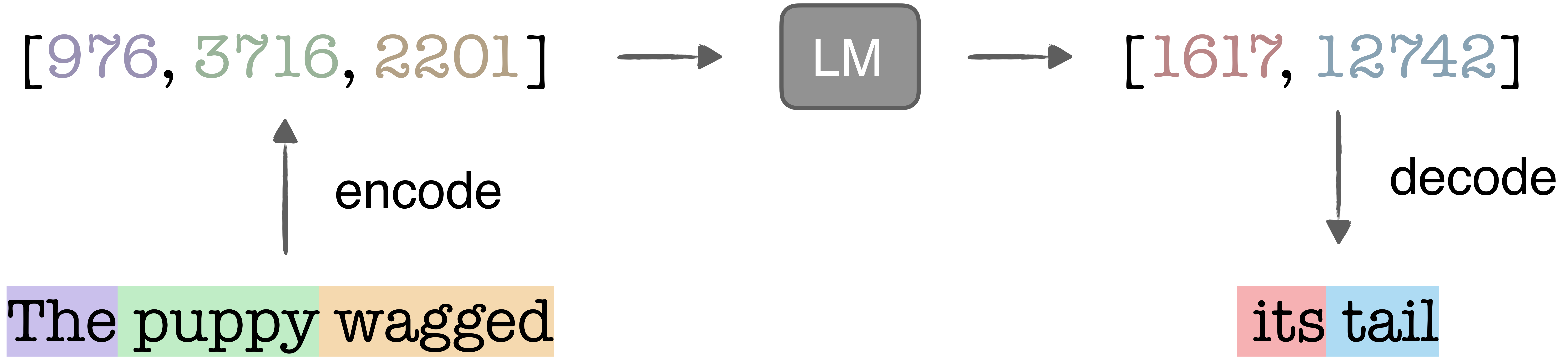
A **tokenizer** translates between text and a sequence of **tokens** that a language model (LM) learns representations over

The **vocabulary** V is the set of known tokens



Word-level tokenization

V = set of all words in the English language



Word-level tokenization

Word-level tokenization

❌ **Cons**

Word-level tokenization

❌ **Cons**

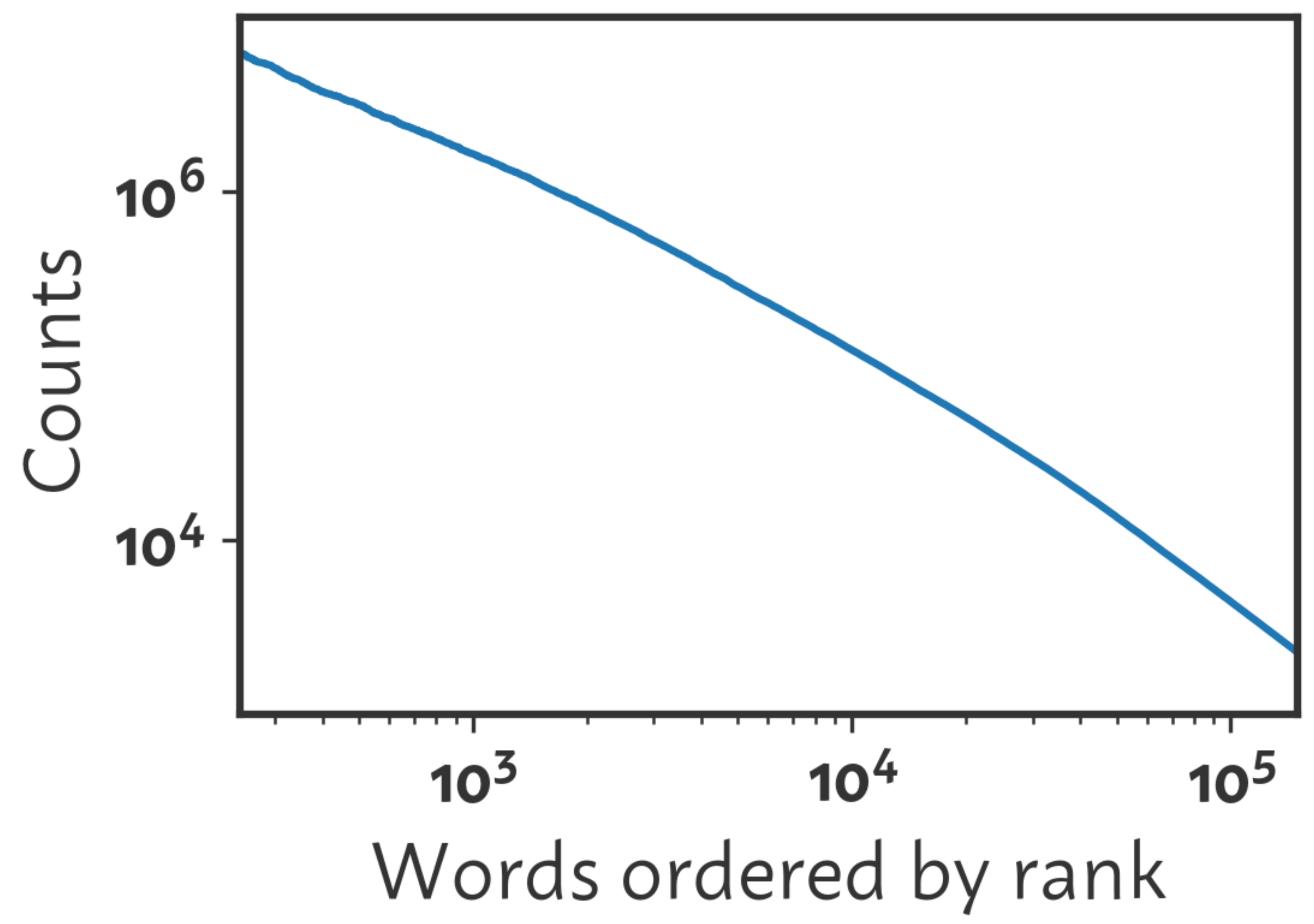
- $|V|$ can be quite large
  - Webster's English dictionary has ~470,000 words!

Word-level tokenization

❌ **Cons**

- $|V|$ can be quite large
  - Webster's English dictionary has ~470,000 words!
- Long tail of infrequent words
  - **Zipf's law**: word freq. is inversely prop. to rank

$$\text{word freq} \propto \frac{1}{\text{word rank}}$$

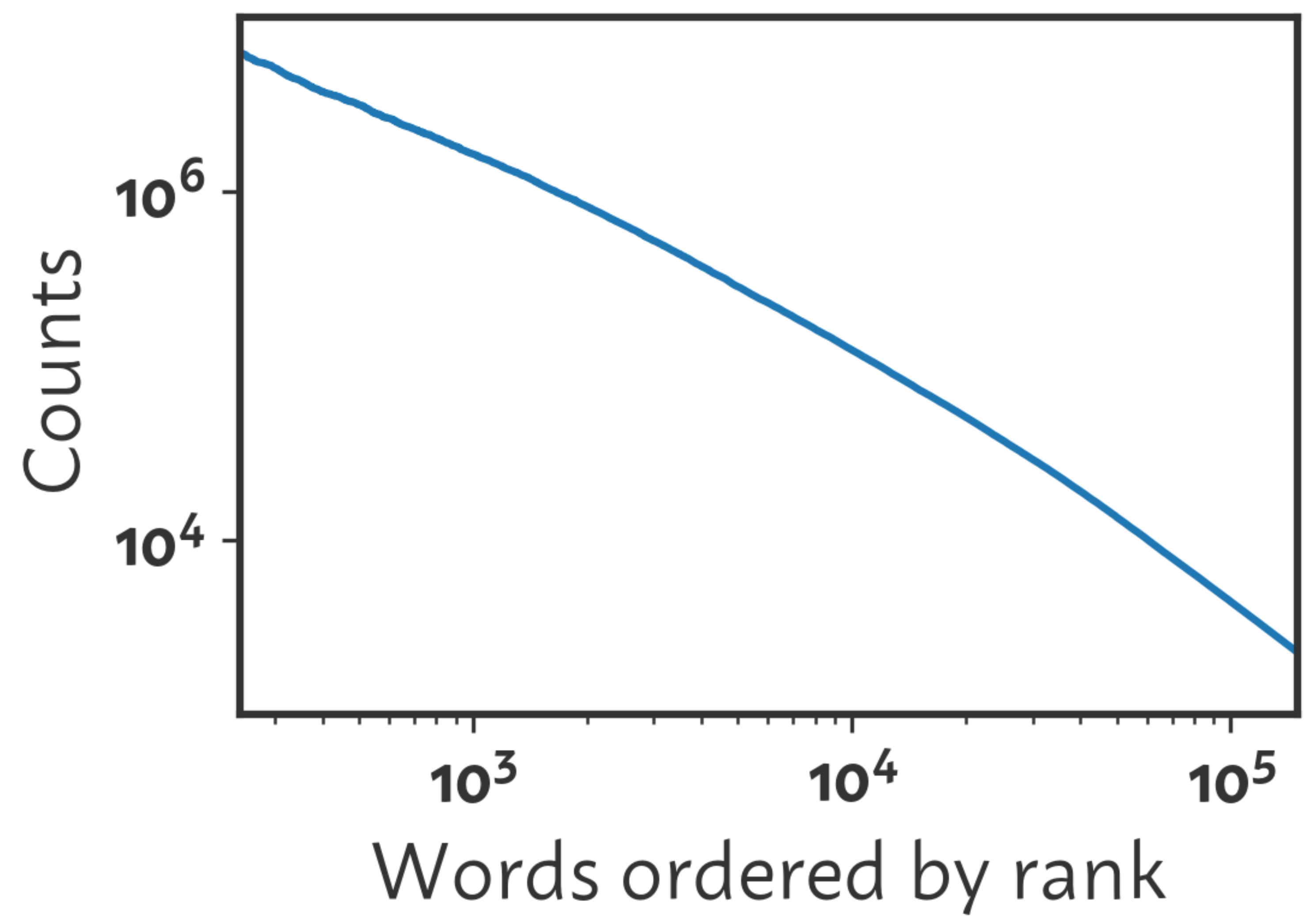


Word-level tokenization

❌ Cons

- $|V|$ can be quite large
  - Webster's English dictionary has ~470,000 words!
- Long tail of infrequent words
  - **Zipf's law**: word freq. is inversely prop. to rank
- Language is changing all the time
  - 690 new words added in Sep 2023: "rizz," "goated," "bussin'," "mid"

$$\text{word freq} \propto \frac{1}{\text{word rank}}$$

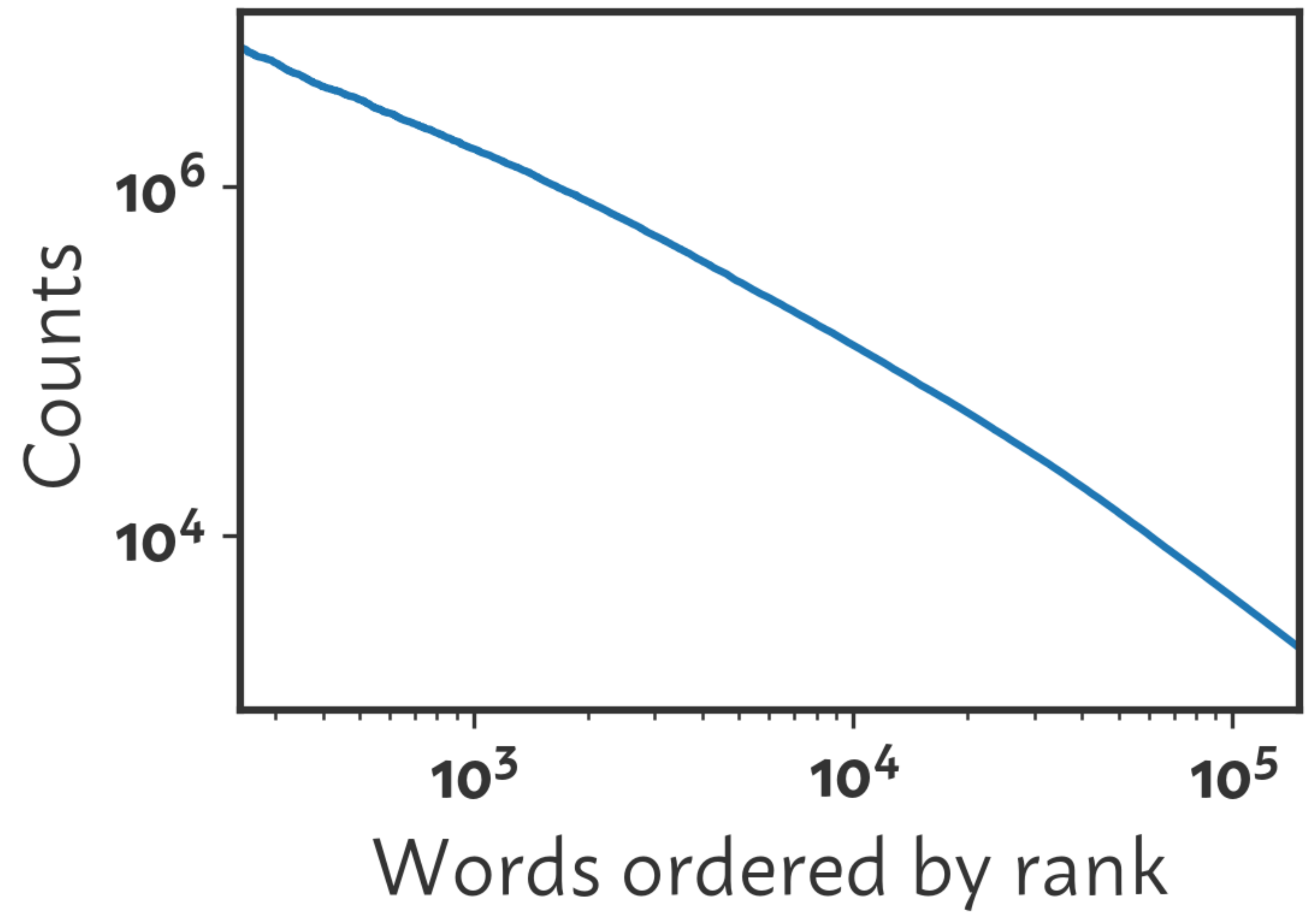


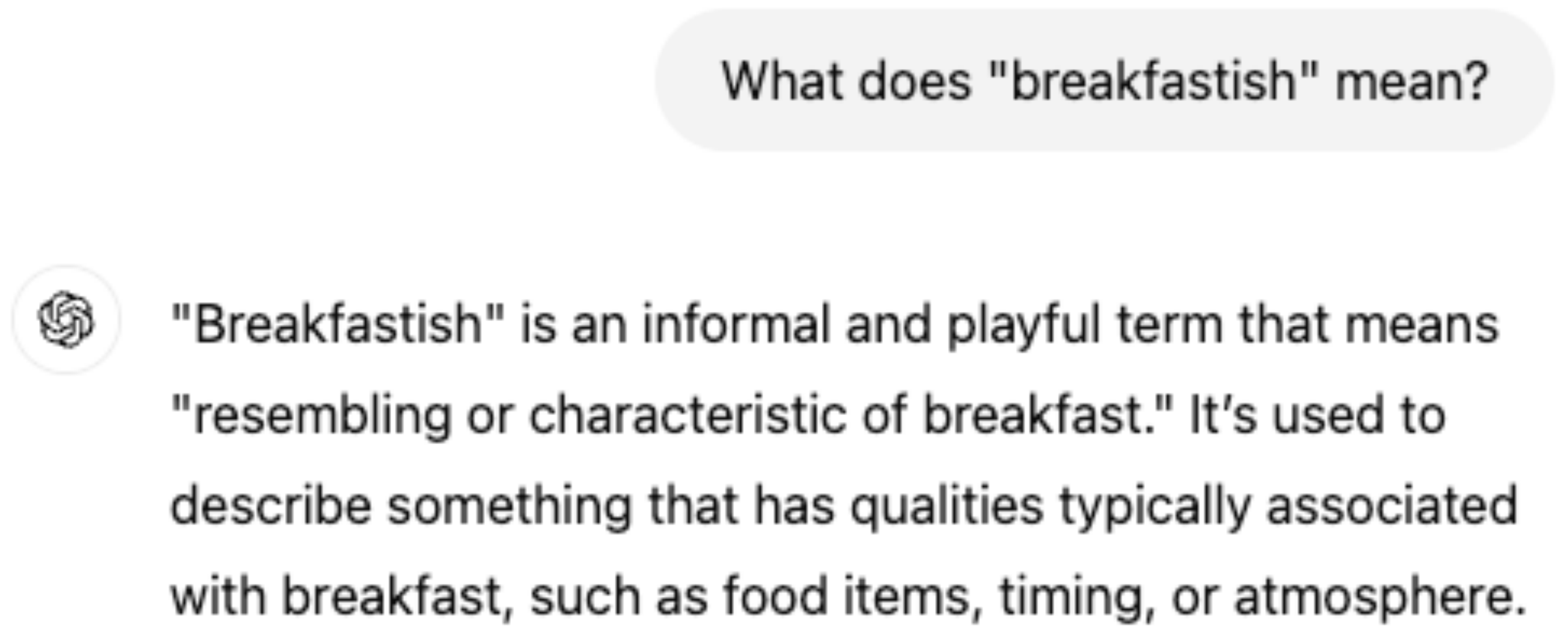
Word-level tokenization

❌ **Cons**

- $|V|$ can be quite large
  - Webster's English dictionary has ~470,000 words!
- Long tail of infrequent words
  - **Zipf's law**: word freq. is inversely prop. to rank
- Language is changing all the time
  - 690 new words [added in Sep 2023](): "rizz," "goated," "bussin'," "mid"
- Still need a way to deal with unknown words

$$\text{word freq} \propto \frac{1}{\text{word rank}}$$





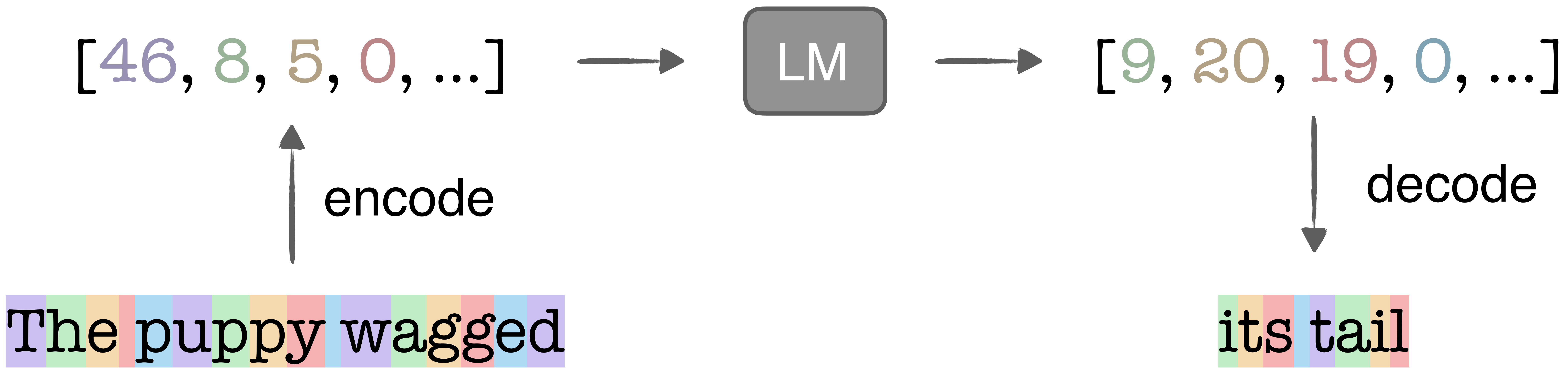
What does "breakfastish" mean?

"Breakfastish" is an informal and playful term that means "resembling or characteristic of breakfast." It's used to describe something that has qualities typically associated with breakfast, such as food items, timing, or atmosphere.

Character-level tokenization

$$V = \{a, b, c, \ldots, z, A, B, C, \ldots, Z\}$$

(plus spaces + punctuation?)



Character-level tokenization

✅ ❌

Character-level tokenization

✅ Pros

❌ Cons

Character-level tokenization

✅ **Pros**

- Small vocabulary size

❌ **Cons**

Character-level tokenization

✅ **Pros**

- Small vocabulary size
- Complete coverage of input

❌ **Cons**

Character-level tokenization

✅ **Pros**

- Small vocabulary size
- Complete coverage of input
- Direct observation of spelling

❌ **Cons**

Character-level tokenization

✅ **Pros**

- Small vocabulary size
- Complete coverage of input
- Direct observation of spelling

❌ **Cons**

- Super long sequences

Character-level tokenization

✅ **Pros**

- Small vocabulary size
- Complete coverage of input
- Direct observation of spelling

❌ **Cons**

- Super long sequences
- Difficult to learn over

Subword tokenization

Subword tokenization

How can we combine the high coverage of character-level representation with the efficiency of word-level representation?

Subword tokenization

How can we combine the <u>high coverage</u> of character-level representation with the <u>efficiency</u> of word-level representation?

Tokens are **subwords**, i.e., *parts* of words

Subword tokenization

How can we combine the high coverage of character-level representation with the efficiency of word-level representation?

Tokens are **subwords**, i.e., *parts* of words

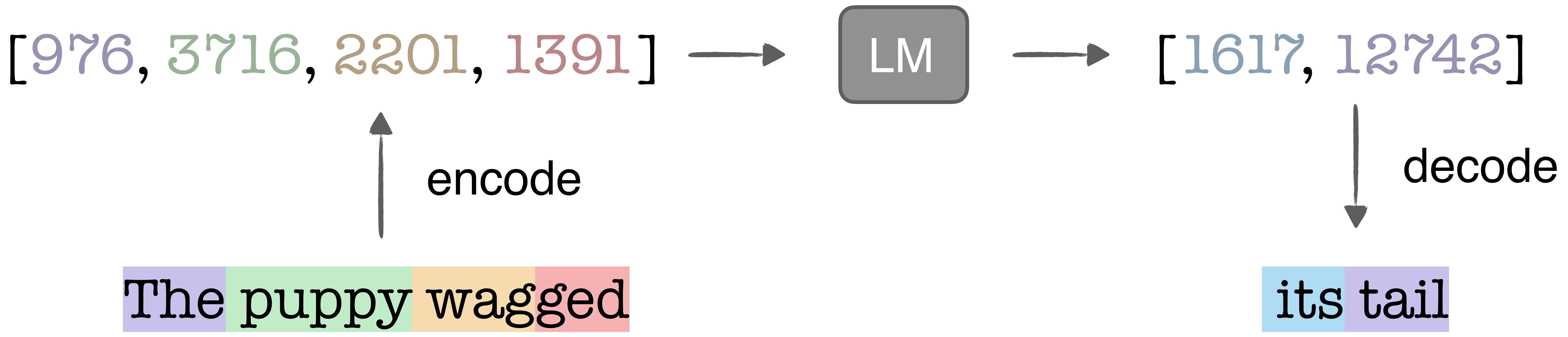
Instead of defining the vocabulary a-priori, use *data* to tell us what our vocabulary should be

Subword tokenization

How can we combine the high coverage of character-level representation with the efficiency of word-level representation?

Tokens are **subwords**, i.e., *parts* of words

Instead of defining the vocabulary a-priori, use *data* to tell us what our vocabulary should be



BPE: Byte Pair Encoding

Universal method today for learning subword tokenizers

Intuition: build the vocabulary bottom-up by repeatedly merging common token sequences into new tokens

Introduced by Sennrich et al., 2016 & popularized by GPT-2 (2019)

BPE Algorithm

Required:

Training data D

Desired vocab size N

Algorithm:

1. Pretokenize D by splitting on whitespace
2. Initialize V as characters in D
3. Convert D into sequence of tokens (i.e., characters)
4. While $|V| < N$:
    a. Get counts of all bigrams (v_i, v_j) in D
    b. Merge most frequent pair 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

BPE Algorithm

Given: Training data D

```
tweetle_beetles_battle
```

BPE Algorithm

1. Pretokenize D by splitting on whitespace

```
 tweetle
_beetles
_battle
```

BPE Algorithm

1. Pretokenize D by splitting on whitespace

```
 tweetle
_beetles
_battle
```

BPE Algorithm

2. Initialize V as characters in D

```
 tweetle
_beetles
_battle
```

BPE Algorithm

3. Convert D into sequence of tokens (i.e., characters)

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e          t w     1
_ b e e t l e s
_ b a t t l e
```

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

$\texttt{t}\ \boxed{\texttt{w}\ \texttt{e}}\ \texttt{e}\ \texttt{t}\ \texttt{l}\ \texttt{e}$

_ b e e t l e s

_ b a t t l e

t w	1
w e	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

t w $\boxed{\text{e e}}$ t l e

_ b e e t l e s

_ b a t t l e

t w	1
w e	1
e e	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1
w e	1
e e	1
e t	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1
w e	1
e e	1
e t	1
t l	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1
w e	1
e e	1
e t	1
t l	1
l e	1

BPE Algorithm

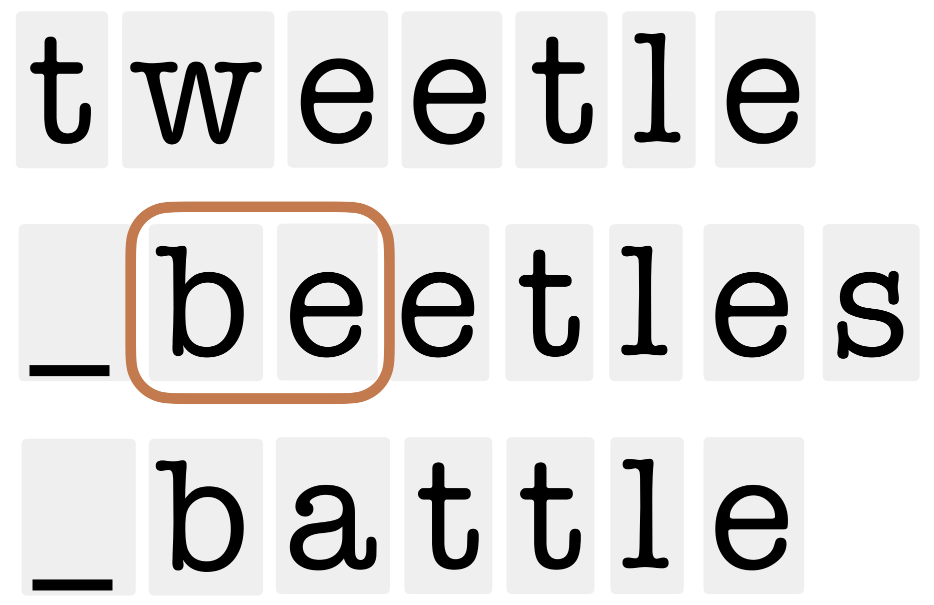
4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	1
w e	1		
e e	1		
e t	1		
t l	1		
l e	1		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D



t w	1	_ b	1
w e	1	b e	1
e e	1		
e t	1		
t l	1		
l e	1		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	1
w e	1	b e	1
e e	2		
e t	1		
t l	1		
l e	1		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	1
w e	1	b e	1
e e	2		
e t	2		
t l	1		
l e	1		

BPE Algorithm

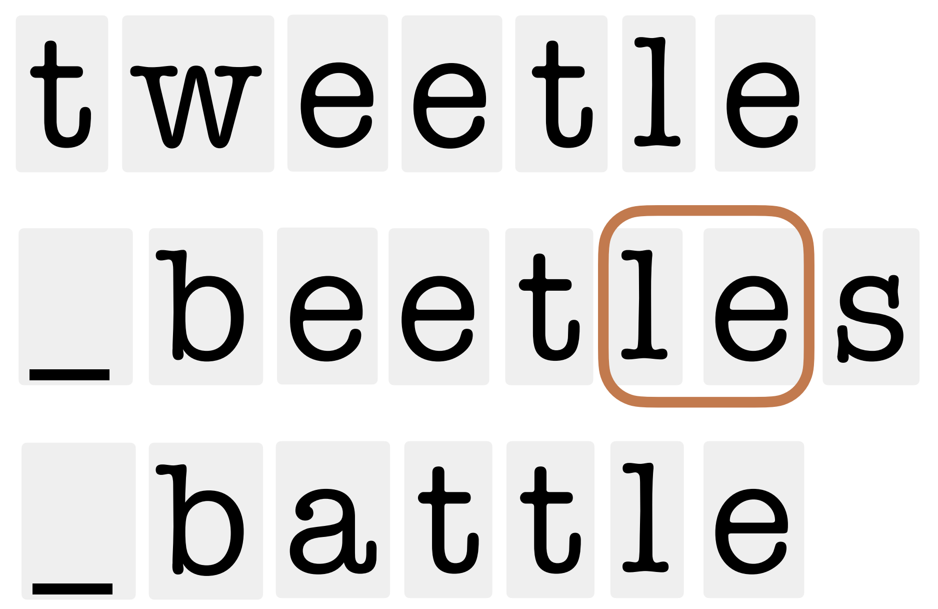
4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	1
w e	1	b e	1
e e	2		
e t	2		
t l	2		
l e	1		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D



t w	1	_ b	1
w e	1	b e	1
e e	2		
e t	2		
t l	2		
l e	2		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	1
w e	1	b e	1
e e	2	e s	1
e t	2		
t l	2		
l e	2		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2		
t l	2		
l e	2		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

t w e e t l e

_ b e e t l e s

_ $\boxed{\texttt{b a}}$ t t l e

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	2		
l e	2		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

t w e e t l e

_ b e e t l e s

_ b $\boxed{\texttt{a t}}$ t l e

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	2	a t	1
l e	2		

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	2	a t	1
l e	2	t t	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

t w e e t l e

_ b e e t l e s

_ b a t $\boxed{\texttt{t l}}$ e

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
l e	2	t t	1

BPE Algorithm

4a. Get counts of all bigrams (v_i, v_j) in D

t w e e t l e

_ b e e t l e s

_ b a t t l e

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
l e	3	t t	1

BPE Algorithm

Merge List

1
2
3
4
⋮

Text

t w e e t l e
_ b e e t l e s
_ b a t t l e

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
l e	3	t t	1

BPE Algorithm

Merge List

1
2
3
4
⋮

Text

```
t w e e t l e
_ b e e t l e s
_ b a t t l e
```

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
l e	3	t t	1

4b. Find most frequent pair (v_i, v_j)

BPE Algorithm

Merge List

1
2
3
4
⋮

Text

t w e e t l e
_ b e e t l e s
_ b a t t l e

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
$\boxed{\text{l e} \quad 3}$		t t	1

4b. Find most frequent pair (v_i, v_j)

BPE Algorithm

Merge List

1 l e
2
3
4

⋮

add to merge list

Text

t w e e t l e
_ b e e t l e s
_ b a t t l e

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
$\boxed{\text{l e} \quad 3}$		t t	1

4b. Find most frequent pair (v_i, v_j)

BPE Algorithm

Merge List

1	l e
2	
3	
4	

⋮

Text

t w e e t le

_ b e e t le s

_ b a t t le

4c. Replace all instances of $v_i v_j$ in D with v_n

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	e s	1
e t	2	b a	1
t l	3	a t	1
$\boxed{\text{l e} \quad 3}$		t t	1

BPE Algorithm

Merge List

1 l e

2

3

4

⋮

Text

t w e e t le

_ b e e t le s

_ b a t t le

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
t le	3	a t	1
		t t	1

4a. Update pair frequencies

BPE Algorithm

Merge List

1 l e
2
3
4

⋮

Text

t w e e t le
_ b e e t le s
_ b a t t le

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
t le	3	a t	1
		t t	1

BPE Algorithm

Merge List

1 l e

2

3

4

⋮

Text

t w e e t le

_ b e e t le s

_ b a t t le

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
t le	3	a t	1
		t t	1

4b. Find most frequent pair

BPE Algorithm

Merge List

1 l e
2
3
4

⋮

Text

t w e e t le
_ b e e t le s
_ b a t t le

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
$\boxed{\text{t le} \quad 3}$		a t	1
		t t	1

4b. Find most frequent pair

BPE Algorithm

Merge List

1	l e
2	t le
3	
4	

add to merge list

⋮

Text

t w e e t le

_ b e e t le s

_ b a t t le

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
t le	3	a t	1
		t t	1

4b. Find most frequent pair

BPE Algorithm

Merge List

1	l e
2	t le
3	
4	

⋮

Text

t w e e tle
_ b e e tle s
_ b a t tle

4c. Apply merge to text

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	le s	1
e t	2	b a	1
t le	3	a t	1
		t t	1

BPE Algorithm

Merge List

1	l e
2	t le
3	
4	

⋮

Text

t w e e tle
_ b e e tle s
_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
e tle	2	b a	1
		a t	1
		t tle	1

4a. Update pair frequencies

BPE Algorithm

Merge List

1 l e
2 t le
3
4

⋮

Text

t w e e tle
_ b e e tle s
_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
e tle	2	b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1 l e
2 t le
3
4

⋮

Text

t w e e tle
_ b e e tle s
_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
e tle	2	b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	
4	
	⋮

Text

t w e e tle

_ b e e tle s

_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
$\boxed{\text{e tle} \quad 2}$		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l	e
2	t	le
3	e	tle
4		

⋮

Text

`t` `w` `e` `e` `tle`
`_` `b` `e` `e` `tle` `s`
`_` `b` `a` `t` `tle`

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
e tle	2	b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	

⋮

Text

t w e etle
_ b e etle s
_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e e	2	tle s	1
e tle	2	b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	

⋮

Text

t w e etle

_ b e etle s

_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e etle	2	etle s	1
		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	

⋮

Text

t w e etle

_ b e etle s

_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e etle	2	etle s	1
		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	

⋮

Text

```
t w e etle
_ b e etle s
_ b a t tle
```

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e etle	2	etle s	1
		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	

⋮

Text

t w e etle

_ b e etle s

_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
e etle	2	etle s	1
		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l	e
2	t	le
3	e	tle
4	e	etle

⋮

Text

t w e etle
_ b e etle s
_ b a t tle

Pair Frequencies

t w	1		_ b	2
w e	1		b e	1
e etle	2		etle s	1
			b a	1
			a t	1
			t tle	1

BPE Algorithm

Merge List

1	l e
2	t le
3	e tle
4	e etle
	⋮

Text

t w eetle

_ b eetle s

_ b a t tle

Pair Frequencies

t w	1	_ b	2
w e	1	b e	1
$\boxed{\text{e etle 2}}$		etle s	1
		b a	1
		a t	1
		t tle	1

BPE Algorithm

Merge List

1	l	e
2	t	le
3	e	tle
4	e	etle

⋮

Text

`t` `w` `eetle`
`_` `b` `eetle` `s`
`_` `b` `a` `t` `tle`

Pair Frequencies

t w	1
w e	1
_ b	2
b e	1
eetle s	1
b a	1
a t	1
t tle	1

… until we reach the desired vocabulary size, $|V| = N$

BPE Algorithm

To tokenize new text at test time, we split it into the characters and apply merge rules in order.

Merge List

1	`l` `e`
2	`t` `le`
3	`e` `tle`
4	`e` `etle`
	⋮

BPE Algorithm

To tokenize new text at test time, we split it into the characters and apply merge rules in order.

Merge List

1	l	e
2	t	le
3	e	tle
4	e	etle

⋮

$$\begin{tikzcd}
\text{w a t t l e} \arrow[d, bend right] \\
\text{w a t t le} \arrow[d, bend right] \\
\text{w a t tle}
\end{tikzcd}$$

BPE: Examples

Given this BPE tokenizer, how would `_the` be tokenized?

Vocab

`t`

`h`

`e`

`_t`

`_th`

`he`

Merge List

`_ t`

`_t h`

`h e`

BPE: Examples

Given this BPE tokenizer, how would _the be tokenized?

Vocab

- `t`
- `h`
- `e`
- `_t`
- `_th`
- `he`

Merge List

- `_ t`
- `_t h`
- `h e`

Answer:

$$\begin{tikzcd}
\texttt{\_ t h e} \arrow[d, bend right] \\
\texttt{\_t h e} \arrow[d, bend right] \\
\texttt{\_th e}
\end{tikzcd}$$

BPE: Examples

Given this BPE tokenizer, how would `_the` be tokenized?

Vocab	Merge List
`t`	
`h`	
`e`	
`_t`	`_ t`
`he`	`h e`
`_th`	`_t h`

BPE: Examples

Given this BPE tokenizer, how would `_the` be tokenized?

Vocab

- `t`
- `h`
- `e`
- `_t`
- `he`
- `_th`

Merge List

- `_ t`
- `h e`
- `_t h`

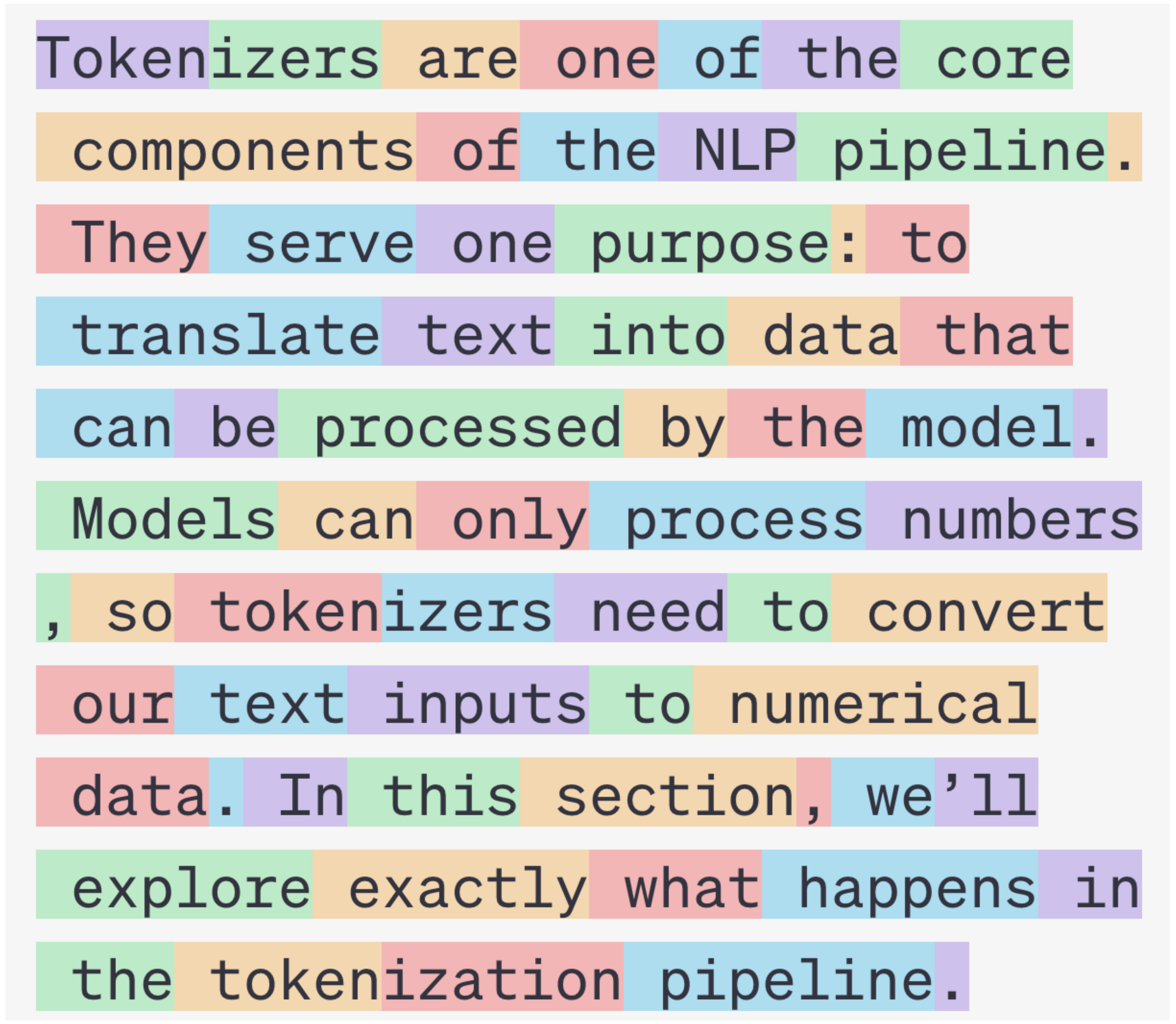
Answer:

$$\begin{tikzcd}
\texttt{\_ t h e} \arrow[d, bend right] \\
\texttt{\_ t he} \arrow[d, bend right] \\
\texttt{\_t he}
\end{tikzcd}$$

ChatGPT’s tokenizer

```
Tokenizers are one of the core components of the NLP
pipeline. They serve one purpose: to translate text
into data that can be processed by the model. Models
can only process numbers, so tokenizers need to
convert our text inputs to numerical data. In this
section, we’ll explore exactly what happens in the
tokenization pipeline.
```

https://platform.openai.com/tokenizer



Tokenizers are one of the core
components of the NLP pipeline.
They serve one purpose: to
translate text into data that
can be processed by the model.
Models can only process numbers
, so tokenizers need to convert
our text inputs to numerical
data. In this section, we’ll
explore exactly what happens in
the tokenization pipeline.

Subword tokenizers

Subword tokenizers

✅ Pros

❌ Cons

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

❌ Cons

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

Some shared representations

wagged

❌ Cons

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

Some shared representations

wagged

❌ Cons

No association between related words

Run ≠ run ≠ RUN

_Hello ≠ Hello

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

Some shared representations

wagged

❌ Cons

No association between related words

Run ≠ run ≠ RUN

_Hello ≠ Hello

Learn the good, bad, & ugly in data

GPT-2 tokens[1]: _RandomRedditor, _SolidGoldMagikarp, PsyNetMessage

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

Some shared representations

wagged

❌ Cons

No association between related words

Run ≠ run ≠ RUN

_Hello ≠ Hello

Learn the good, bad, & ugly in data

GPT-2 tokens[1]: _RandomRedditor, _SolidGoldMagikarp, PsyNetMessage

No direct observation of spelling

Subword tokenizers

✅ Pros

Everything can be represented with the vocabulary

Some shared representations

`wagged`

❌ Cons

No association between related words

`Run` ≠ `run` ≠ `RUN`

`_Hello` ≠ `Hello`

Learn the good, bad, & ugly in data

GPT-2 tokens[1]: `_RandomRedditor`, `_SolidGoldMagikarp`, `PsyNetMessage`

No direct observation of spelling

“Intermediate” tokens can be useless

`entucky` token is completely subsumed by `_Kentucky`

What could we do differently?

Variant: how to treat whitespace

Variant: how to treat whitespace

Instead of merging spaces into the beginning of words, use special "continue word" character

- With whitespace:[`_Token`, `ization`, `_is`, `_cool`]
- W/o whitespace: [`Token`, `##ization`, `is`, `cool`]

Variant: how to treat whitespace

Instead of merging spaces into the beginning of words, use special "continue word" character

With whitespace:[`_Token`, `ization`, `_is`, `_cool`]

W/o whitespace: [`Token`, `##ization`, `is`, `cool`]

❌ **Cons**

Variant: how to treat whitespace

Instead of merging spaces into the beginning of words, use special “continue word” character

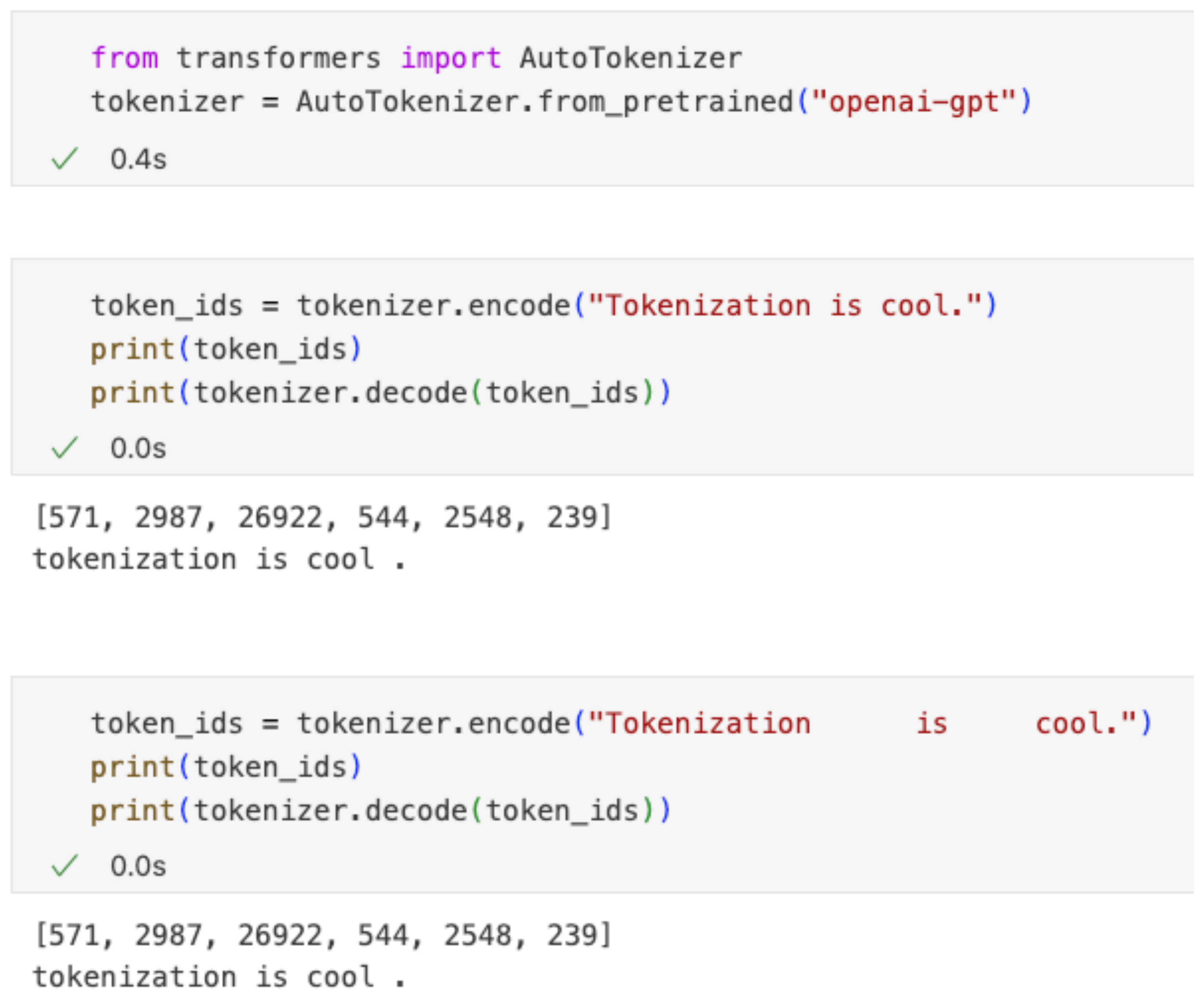
With whitespace:[_Token, ization, _is, _cool]

W/o whitespace: [Token, ##ization, is, cool]

✗ Cons

Loses whitespace information

(especially problematic for code!)



```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("openai-gpt")
✓ 0.4s
```

```
token_ids = tokenizer.encode("Tokenization is cool.")
print(token_ids)
print(tokenizer.decode(token_ids))
✓ 0.0s
```

```
[571, 2987, 26922, 544, 2548, 239]
tokenization is cool .
```

```
token_ids = tokenizer.encode("Tokenization      is      cool.")
print(token_ids)
print(tokenizer.decode(token_ids))
✓ 0.0s
```

```
[571, 2987, 26922, 544, 2548, 239]
tokenization is cool .
```

Variant: byte-based

Variant: byte-based

Originally, we presented BPE as having characters as the smallest unit

Variant: byte-based

Originally, we presented BPE as having characters as the smallest unit

But there are *many* characters if you want to support…

- Character-based languages (e.g., שمঅ学ひ한กบЖ)
- Non-alphanumeric characters (e.g., 💀🫠😍)

Variant: byte-based

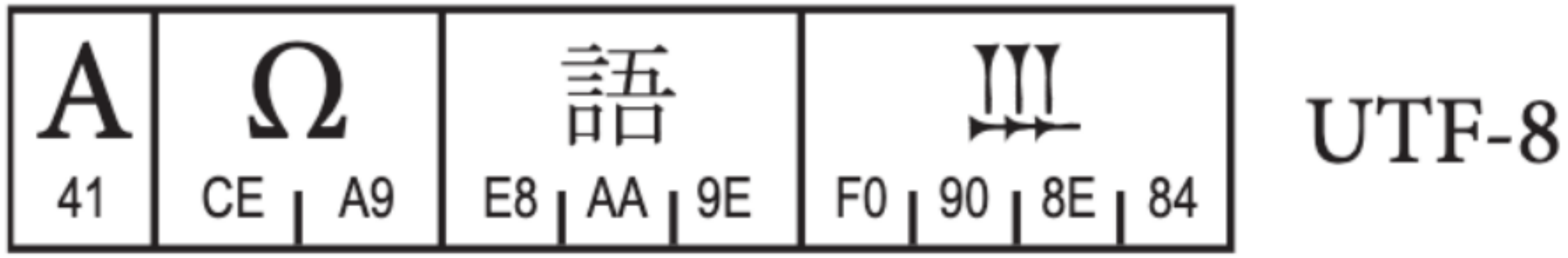
Originally, we presented BPE as having characters as the smallest unit

But there are *many* characters if you want to support…

- Character-based languages (e.g., שمঅ学ひ한กՍЖ)
- Non-alphanumeric characters (e.g., 💀🫠😍)

Instead, use UTF-8 to map all characters in Unicode to byte strings (of 1-4 bytes)

Initialize base vocab as the set of 256 bytes, instead of the English characters

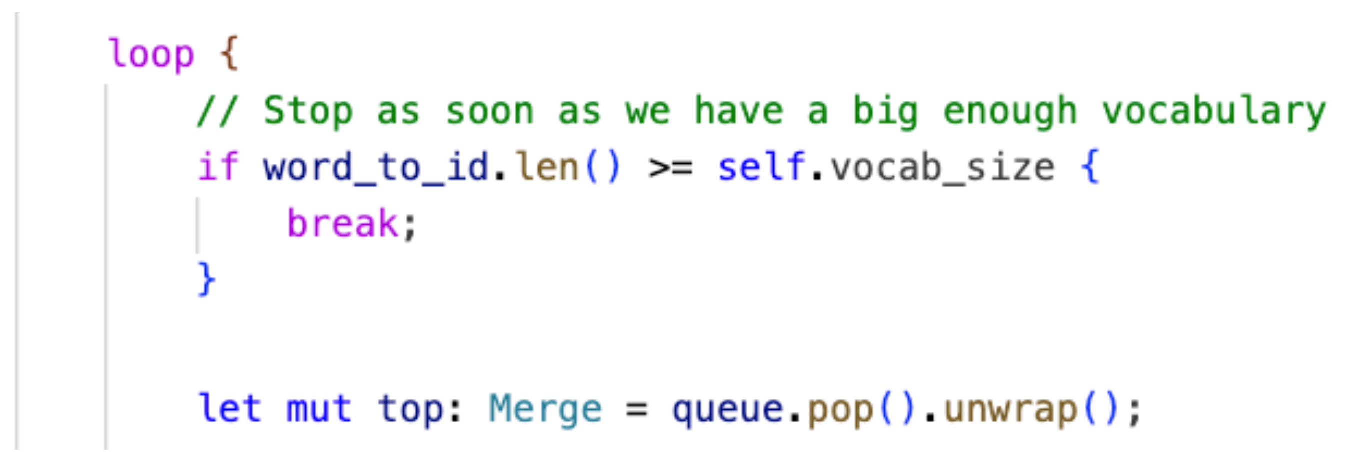


Variants: pretokenization decisions

Recall: pretokenization sets limits on what boundaries our tokens can cross

How should we pretokenize…

- Digits? Consider: 10 vs. 1000000 vs. 5493747
- Consecutive spaces? Consider:



```
loop {
    // Stop as soon as we have a big enough vocabulary
    if word_to_id.len() >= self.vocab_size {
        break;
    }

    let mut top: Merge = queue.pop().unwrap();
```

- Punctuation? Consider: `yay!`, `!=`, `get.`, `.get`
- Newlines? Consider: `;\n`
- Whitespace? Consider: `thank you`, `New York`