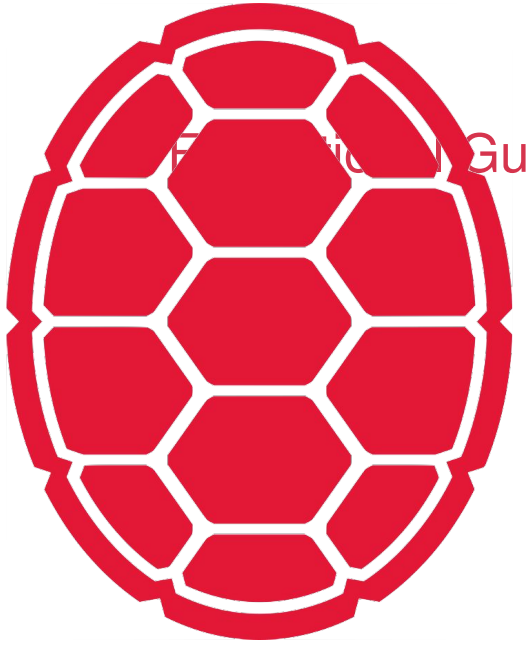
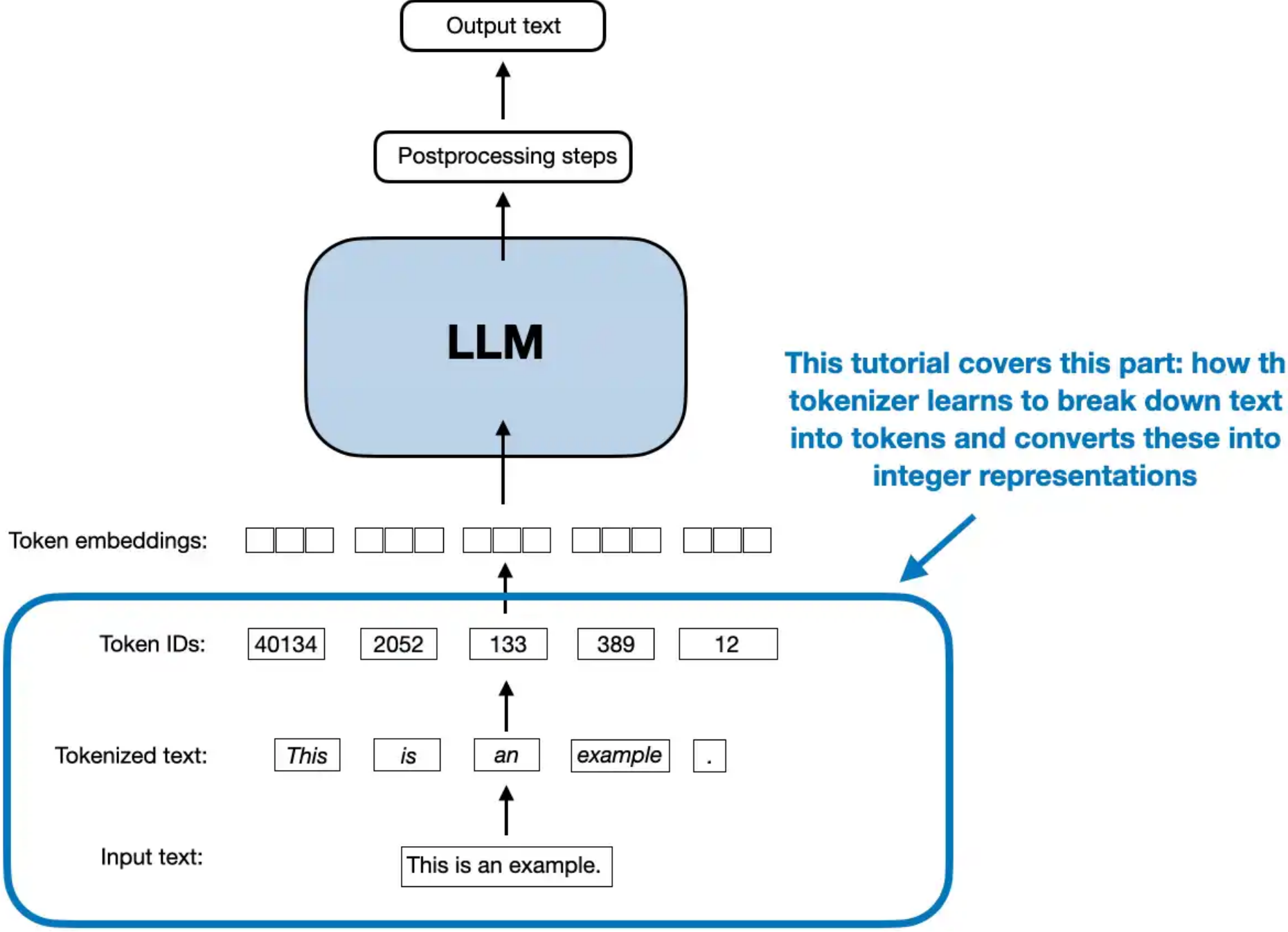


# Tokenization

## [illegible] Guide with Python Code for BPE Tokenization

University of Maryland

Slides and code adapted from Sebastian Raschka `https://sebastianraschka.com/blog/2025/bpe-from-scratch.html`




Strings to Token IDs in Muppet Model

# History

- “A New Algorithm for Data Compression” –Philip G. Gage (BS from UC Boulder)
- Popularized for nlp by Sennrich et al., 2016

**Neural Machine Translation of Rare Words with Subword Units**

**Rico Sennrich** and **Barry Haddow** and **Alexandra Birch**
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk, bhaddow@inf.ed.ac.uk

# Why Use Subword Tokenization Like BPE?

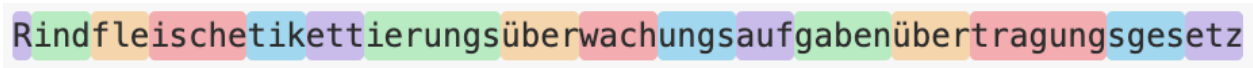
- Single byte tokens lead to long token sequences
- BPE merges frequent byte pairs into subwords
- Most frequent words will get their own tokens

# Why Use Subword Tokenization Like BPE?

- Single byte tokens lead to long token sequences
- BPE merges frequent byte pairs into subwords
- Most frequent words will get their own tokens
- But still allows you to (re)create arbitrary words:

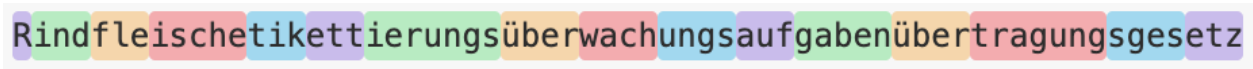
# Why Use Subword Tokenization Like BPE?

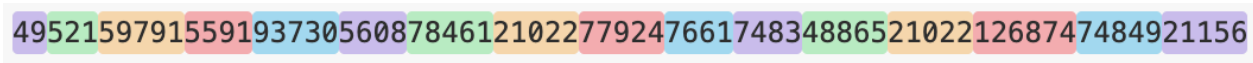
- Single byte tokens lead to long token sequences
- BPE merges frequent byte pairs into subwords
- Most frequent words will get their own tokens
- But still allows you to (re)create arbitrary words:



# Why Use Subword Tokenization Like BPE?

- Single byte tokens lead to long token sequences
- BPE merges frequent byte pairs into subwords
- Most frequent words will get their own tokens
- But still allows you to (re)create arbitrary words:



# Is it linguistically motivated?

- Kinda: Finds suffixes and prefixes
  - professional·ize
  - re·form
  - un·bel·iev·able

# Is it linguistically motivated?

- Kinda: Finds suffixes and prefixes
  - professional·ize
  - re·form
  - un·bel·iev·able
- Handles mispellings somewhat elegantly

# Is it linguistically motivated?

- Kinda: Finds suffixes and prefixes
  - professional·ize
  - re·form
  - un·bel·iev·able
- Handles mispellings somewhat elegantly
- Does silly stuff
  - Ge·orgetown· University
  - st·raw·berry (n.b. ‘r’ moved between GPT 3.5 and 4)
  - Solid·GoldMagikarp

# GPT-2 Tokenizer Example Using tiktoken

- Use OpenAI's `tiktoken` library for GPT-2 tokenization

```
import tiktoken
gpt2_tokenizer = tiktoken.get_encoding("gpt2")
print(gpt2_tokenizer.encode("This is some text"))
# Output: [1212, 318, 617, 2420]
```

- Reduces 17 characters to 4 tokens
- spaces get attached to following word

# Byte Values and Token Vocabulary Initialization

- Start with all single bytes: 256 possible values (0-255)
  - Includes all single character UTF-8 (ASCII)
  - And all multicharacter UTF-8 starters
- Vocabulary grows by merging frequent byte pairs

# Byte Pair Encoding (BPE) Algorithm Outline

**Algorithm 1** Iterative Greedy BPE (slow).
**Inputs**: sequence $\boldsymbol{x}$, merge count $M$
**Output**: merge sequence $\boldsymbol{\mu}$, tokenized sequence $\boldsymbol{x}$
PAIRFREQ are non-overlapping pair frequencies

$\boldsymbol{\mu} \leftarrow \langle\rangle$
**for** $i$ in $\{0, \ldots, M\}$ **do**
  $\mu \leftarrow \underset{(\mu', \mu'') \in \text{set}(\boldsymbol{x})^2}{\text{argmax}} \text{PAIRFREQ}(\boldsymbol{x}, (\mu', \mu''))$
  $\boldsymbol{x} \leftarrow \text{APPLY}(\mu, \boldsymbol{x})$
  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} \circ \langle \mu \rangle$
**end for**
**return** $\boldsymbol{\mu}, \boldsymbol{x}$

From Zouhar et al., 2023

1. Step 1: Identify the most frequent adjacent pairs
2. Step 2: Replace pairs with new tokens (IDs)
3. Record merges in lookup table (vocabulary)
4. Repeat until no pairs occur more than once (or max vocab size reached)

# BPE Example: Training - Iteration 1

- Text: “the cat in the hat”
- Frequent pair: “th” appears twice.
- Replace “th” with new token ID 256.
- New text: `{256}e cat in {256}e hat`
- Vocabulary: 256 → “th”

# BPE Example: Training - Iteration 2

- Text: `{256}e cat in {256}e hat`
- Frequent pair: “{256}e” appears twice
- Replace with token ID 257.
- New text: `{257} cat in {257} hat`
- Vocabulary updated with 257 → “{256}e”

# BPE Example: Training - Iteration 3

- Text: `{257} cat in {257} hat`
- Frequent pair: “{257} ” (token 257 plus space).
- Replace with token ID 258.
- New text: `{258}cat in {258}hat`
- Vocabulary updated with 258 → “{257} ”

# BPE Example: Training - Iteration 4

- Text: `{258}cat in {258}hat`
- Frequent pair: “at”
- Replace with token ID 259.
- New text: `{258}c{259} in {258}h{259}`
- Vocabulary updated with 259 → “at”

# Encoding

1. You get a stream of tokens (initially your raw bytes)
2. Extract all of the pairs of bytes
3. Find the one that corresponds to the lowest rank (token index)
4. Replace that with token
5. Repeat until no pairs in vocab left
6. Repeats the process / order used in training

- `that`
- `{256}at`
- `{256}{259}`

# Building the Vocabulary

```
for (p0, p1), new_id in self.bpe_merges.items():
  merged_token = self.vocab[p0] + self.vocab[p1]
  self.vocab[new_id] = merged_token
  self.inverse_vocab[merged_token] = new_id
```

- Don't want to go through three token pairs to find "the "
- So map tokens directly to strings (and *vice versa*)

# BPE Decoding

- Compressed text: `{257}{259}{101}{104}`

# BPE Decoding

- Compressed text: `{257}{259}{101}{104}`
- Substitute 257 → “the”: `the{259}{101}{104}`

# BPE Decoding

- Compressed text: `{257}{259}{101}{104}`
- Substitute 257 → “the”: `the{259}{101}{104}`
- Substitute 259 → “at”: `theat{101}{104}`

# BPE Decoding

- Compressed text: `{257}{259}{101}{104}`
- Substitute 257 → “the”: `the{259}{101}{104}`
- Substitute 259 → “at”: `theat{101}{104}`
- Substitute 101 → “e”: `theate{104}`

# BPE Decoding

- Compressed text: `{257}{259}{101}{104}`
- Substitute 257 → “the”: `the{259}{101}{104}`
- Substitute 259 → “at”: `theat{101}{104}`
- Substitute 101 → “e”: `theate{104}`
- Substitute 114 → “r”: `theater`

## BPE Merge

You can shortcut multiple tokens by recording the yield of each token (after you built the vocabulary)

# BPE Training: Preprocessing and Initialization

- Replace spaces with special char “Ġ” (GPT-2 style)
- Initialize vocab with all single bytes (0-255)
- Add unique additional chars from text (e.g., add Greek alphabet if you know you’re going to have Greek text)
- Include allowed special tokens if any (e.g., tag to mark end of generation)
- Convert text to token ID list for merges

# Encoding Text with BPE Tokenizer

- Handles special tokens with regex matching
- Splits text by special tokens, encodes separately
- Encodes known tokens directly
- Tokenizes unknown tokens recursively with BPE

# Recap

- Alternative to WordPiece: Instead of using $p(a, b)$, uses $\frac{p(a,b)}{p(a)p(b)}$
- Relatively simple, efficiency can be improved by memoization (Zouhar et al., 2023)
- Breaks down unknown works (rather than unk)
- Works relatievly well for English
- Issues for lower resource languages, low frequency tokens, and mispellings