Tokenization

CS685 Spring 2025

Advanced Natural Language Processing

Haw-Shiuan Chang

College of Information and Computer Sciences

University of Massachusetts Amherst

Most slides come from Mohit Iyyer

Tokenization

- How do we represent an input text?
- So far in this class… we chop it up into *words*

Input text: students opened their books

Tokenization

- How do we represent an input text?
- So far in this class… we chop it up into *words*

Input text: students opened their books

Input token IDs: 11 298 34 567

This tokenization step requires an external *tokenizer* to detect word boundaries!

Word tokenization

- Not as simple as split on whitespace and punctuation…

Mr. **O'Neill** thinks that the boys' stories about San Francisco **aren't** amusing.

- Word tokenizers require lots of specialized rules about how to handle specific inputs
    - Check out spaCy's tokenizers! (https://spacy.io/)

Handling unknown words

- What happens when we encounter a word at test time that we’ve never seen in our training data?
    - With word level tokenization, we have no way of assigning an index to an unseen word!
    - This means we don’t have a word embedding for that word and thus cannot process the input sequence

Handling unknown words

- What happens when we encounter a word at test time that we've never seen in our training data?
  - With word level tokenization, we have no way of assigning an index to an unseen word!
  - This means we don't have a word embedding for that word and thus cannot process the input sequence
- Solution: replace low-frequency words in training data with a special <UNK> token, use this token to handle unseen words at test time too
  - Why use <UNK> tokens during training?

Limitations of <UNK>

- We lose lots of information about texts with a lot of rare words / entities

```
The chapel is sometimes referred to
as "Hen Gapel Lligwy" ("hen" being
the Welsh word for "old" and "capel"
meaning "chapel").
```

```
The chapel is sometimes referred to
as " Hen <unk> <unk> " (" hen " being
the Welsh word for " old " and "
<unk> " meaning " chapel ").
```

Other limitations

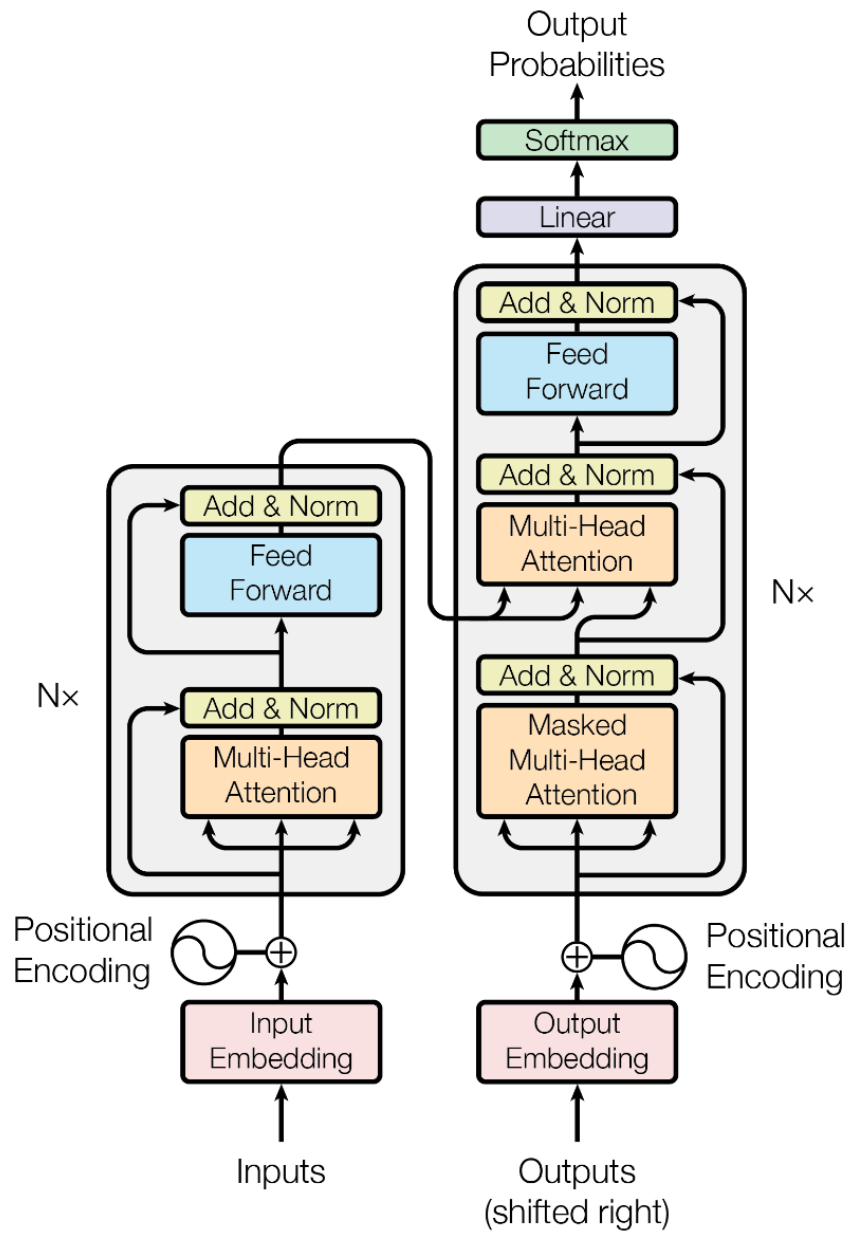
- Word-level tokenization treats different forms of the same word (e.g., “open”, “opened”, “opens”, “opening”, etc) as separate types —> separate embeddings for each

This can be problematic especially when training over smaller datasets, why?

An alternative: character tokenization

- Small vocabulary, just the number of unique characters in the training data!

- However, you pay for this with longer input sequences. Why is this a problem for the models we've discussed?



2016: subword tokenization

- Developed for machine translation by Sennrich et al., ACL 2016

  “The main motivation behind this paper is that the translation of some words is transparent in that they are translatable by a competent translator even if they are novel to him or her, based on a translation of known subword units such as morphemes or phonemes.”

- Later used in BERT, T5, RoBERTa, GPT, etc.

- Relies on a simple algorithm called *byte pair encoding* (Gage, 1994)

Byte pair encoding

- Form base vocabulary (all characters that occur in the training data)

word	frequency
hug	10
pug	5
pun	12
bun	4
hugs	5

- Base vocab: b, g, h, n, p, s, u

Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Now, count up the frequency of each character *pair* in the data, and choose the one that occurs most frequently

word	frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

character pair	frequency
ug	20
pu	17
un	16
hu	15
gs	5

...

Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Now, choose the most common pair (ug) and then merge the characters together into one symbol. Add this new symbol to the vocabulary. Then, retokenize the data

word	frequency
h+*ug*	10
p+*ug*	5
p+u+n	12
b+u+n	4
h+*ug*+s	5

character pair	frequency
un	16
h+ug	15
pu	12
p+ug	5
ug+s	5

...

Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Keep repeating this process! This time we choose *un* to merge, next time we choose *h+ug*, etc.

word	frequency
h+*ug*	10
p+*ug*	5
p+u+n	12
b+u+n	4
h+*ug*+s	5

character pair	frequency
un	16
h+ug	15
pu	12
p+ug	5
ug+s	5

...

Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Eventually, after a fixed number of merge steps, we stop

word	**frequency**
hug	10
p+*ug*	5
p+*un*	12
b+*un*	4
hug + s	5

- new vocab: b, g, h, n, p, s, u, *ug*, *un*, *hug*

Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- To avoid <UNK>, all possible characters / symbols need to be included in the base vocab. This can be a lot if including all unicode characters (there are ~138K unicode symbols)!

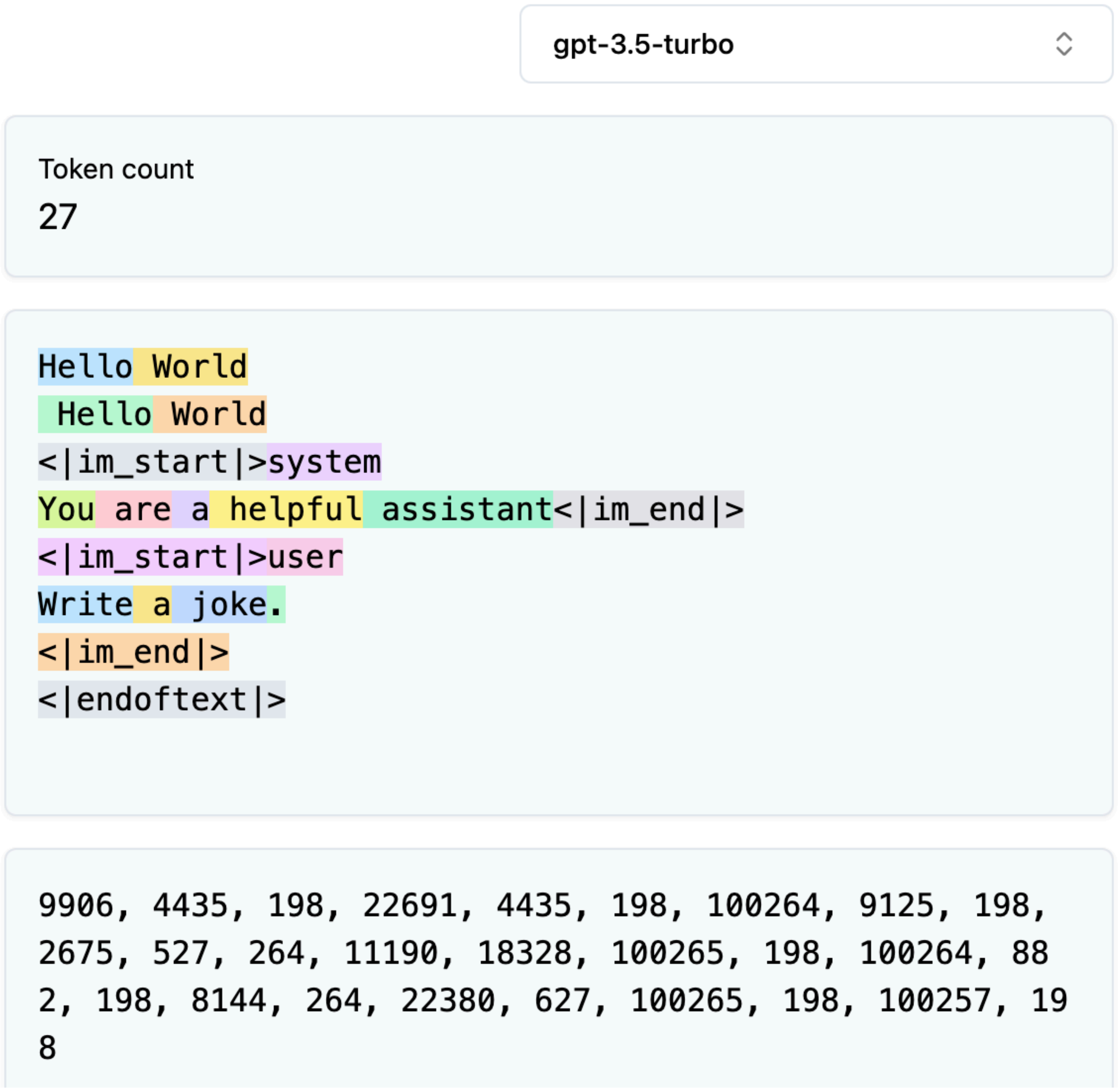
- GPT-2 uses *bytes* as the base vocabulary (size 256) and then applies BPE on top of this sequence (with some rules to prevent certain types of merges).

- Commonly have vocabulary sizes of 32K to 64K

Tokenization in LM

- White Space Prefix
  - “ĠHello"
  - “_Hello”
- Special Tokens

https://tiktokenizer.vercel.app/?model=gpt-3.5-turbo



gpt-3.5-turbo

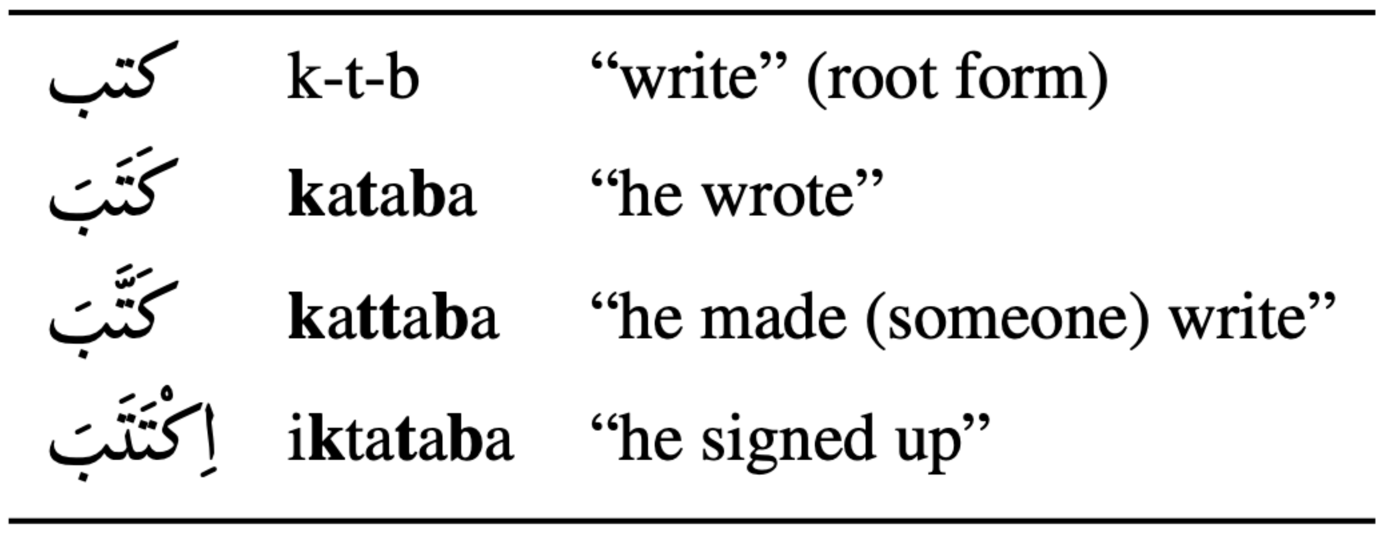
Token count
27

```
Hello World
 Hello World
<|im_start|>system
You are a helpful assistant<|im_end|>
<|im_start|>user
Write a joke.
<|im_end|>
<|endoftext|>
```

```
9906, 4435, 198, 22691, 4435, 198, 100264, 9125, 198,
2675, 527, 264, 11190, 18328, 100265, 198, 100264, 88
2, 198, 8144, 264, 22380, 627, 100265, 198, 100257, 19
8
```

Limitations of subwords

- One word could have multiple tokenization ways
- Hard to apply to languages with agglutinative (e.g., Turkish) or non-concatenative (e.g., Arabic) morphology
- Pretokenization rules don't work on some languages (Thai, Chinese don't use spaces between words; Hawaiian uses punctuation as consonants)



كتب	k-t-b	"write" (root form)
كَتَبَ	**k**a**t**a**b**a	"he wrote"
كَتَّبَ	**k**a**tt**a**b**a	"he made (someone) write"
اِكْتَتَبَ	i**k**ta**t**a**b**a	"he signed up"

Table 1: Non-concatenative morphology in Arabic.[4] The root contains only consonants; when conjugating, vowels, and sometimes consonants, are interleaved with the root. The root is not separable from its inflection via any contiguous split.

Clark et al., 2021, "CANINE"