Tokenization

Negar Foroutan

EPFL

nlp lab

Announcements

- **Guest Lecture Tomorrow:** Kayo Yin (UC Berkeley)
  - **Normal class time:** 13h15 in CE 1 6
- **SCITAS Tutorial Tomorrow:** Daniel Jana (SCITAS) & Zeming Chen (NLP)
  - **Normal exercise session time & place:** 14h15 in CE 1 1
  - Bring laptops — interactive component where you practice commands to launch jobs on SCITAS cluster
- **Assignment 2 grades released next week**
  - Same procedure as for Assignment 1
- **Course Project:** Milestone 1 due **Sunday, May 5th**!
  - **Proposals** submitted by 12 PM on Monday, May 6th time graded within 2 days and feedback given to your mentors

Assignment 2 Grading Review

- **Assignment grades released early in the week (with feedback)**
  - Assignment rubric provided
  - Students review assignment grades, feedback, and rubric
- **Ed Discussion Threads created for each Assignment part**
  - Student make **private** messages about grading errors in relevant threads
  - TAs that graded relevant sections will engage with questions
- **Remaining grading disagreements to be discussed at Grade Review Sessions (May 9, 16)**
  - Students that started discussions on Ed will have priority
  - Ed messages made until **Wed., May 8th, noon** for priority on May 9th; **Wed. May 15th, noon** for priority in May 16th session
- **If disagreements remain, students can fill out Google Form for official re-grading**
  - Full assignment will be re-graded (it can go both ways!)

Lecture’s Outline

- **Tokenization**
  - Definition & Motivation
  - Word Tokenization
  - Character Tokenization
  - Subword Tokenization
  - “Tokenization-free” Byte-level Modeling

Tokenization

- The process of turning a stream of textual data into little pieces (tokens)
  - Words, terms, sentences, symbols, or some other meaningful elements

- **Token:** A sequence of characters that are grouped together as a useful semantic unit for processing

We all love the modern NLP course!

↓ **Tokenization**

```
["We", "all", "love", "the", "modern", "NLP", "course", "!",
"We", "all", "love", "the modern", "NLP course", "!",
"w", "e", "a", "l", "t", "h", "m", "o",…,"!"]
…
```

Why do we need tokenization?

Why do we need tokenization?

- Natural language text is a series of unstructured sequences
- Tokenization breaks the text into chunks of information that can be considered as discrete elements
    - A numerical data structure suitable for machine learning

How should we tokenize text?

How to tokenize?

- Text can be analyzed and generated at many granularities
    - From bytes to multi-word expressions
- Traditionally, NLP models operated over words
    - Splits the data using space and delimiters (e.g., “ ” or “;” or “,”)
    - Still complex!
- So far in the course, we’ve assumed the text was cut into words
    - We’ve seen models that use different tokenization, but haven’t discussed in detail

Word Tokenization

- It takes natural breaks, like pauses in speech or spaces in the text
  - Splits the data into its respective words using delimiters (e.g., " " or ";" or ",")

We all love the modern NLP course!

Tokenization

```
["We", "all", "love", "the", "modern", "NLP", "course", "!"]
```

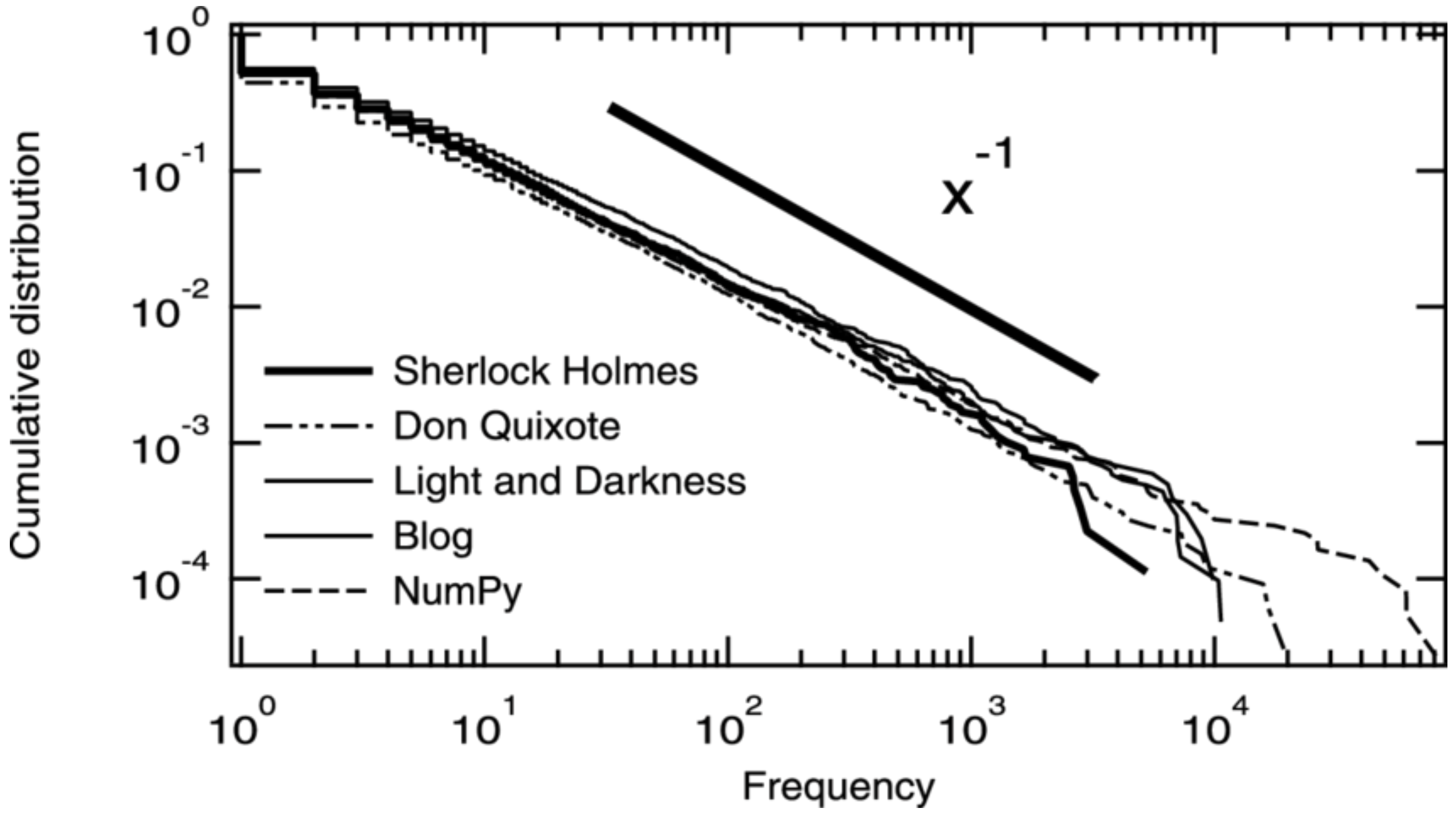
Word Tokenization

- Not as simple as splitting on whitespace and punctuation
  - Ambiguity of the word boundaries
  - Example: Prof. / Dr. / 3-year-old / don’t / 123,456.78
- It requires many specialized rules to handle specific inputs
  - Examples: spaCy and NLTK tokenizers
- Not typically applicable to languages without spaces:
  - Chinese, Japanese, Korean, Thai, Hindi, Tamil, and others

Word Tokenization

- It treats different forms of the same word as separate types
  - (e.g., “talk”, “talks”, “talked”, “talking”, etc)
  - **Problematic when training over smaller datasets**

- Leads to a big vocabulary
  - Zipf’s Law
  - A huge embedding matrix for the input and the output layers
  - Require more computational resources



Word Tokenization

- How to deal with **out-of-vocabulary (OOV)** words?
  - No way of assigning an index to an unseen word
  - No word embedding for that word and cannot process the input sequence
- Replace low-frequency words in training data with a special **<UNK>** token
  - Use this token to handle unseen words at test time too
  - We lose lots of information about texts with a lot of rare words/entities

What else could we do?

Character Tokenization

- Split the raw text into individual characters

We all love the modern NLP course!

Tokenization

[“W”, “e”, “a”, “l”, “l”, “l”, “o”, “v”, “e”, “t”, “h”, ”e”
“m”, “o”, “d”, “e”, “r” “n”, “N”, “L”, “P”,
“c“, “o”, “u”, “r”, “s”, “e”, “!”]

Character Tokenization

- Small vocabulary:
    - The **number of unique characters** in the training data

- Solves the OOV words problem

- More robust to noise and out-of-distribution data

Character Tokenization

- Small vocabulary:
  - The **number of unique characters** in the training data
- Solves the OOV words problem
- More robust to noise and out-of-distribution data
- length of the input increases rapidly
  - **Slower** training and inference
  - Challenging to learn the relationship between the characters to form meaningful words

What else could we do?

Subword Tokenization

- A solution between word- and character-based tokenization
  - Splits the text into subwords (or n-gram characters)

$\boxed{\text{We all love the modern NLP course!}}$

Tokenization ↓

```
[“We”, “al”, “l”, “lo”, “ve”, “the”, “modern”, “NL”, “P”, “course”, “!”]
```

Subword Tokenization

- Uses the following principles:
  - Frequently used words should not be split into smaller subwords
  - Rare words should be decomposed into meaningful subwords
- These methods have two parts:
  - A token learner that takes a raw training corpus and induces a vocabulary (a set of tokens)
  - A token segmenter that takes a raw test sentence and tokenizes it according to that vocabulary

Subword Tokenization

- More meaningful individual tokens
- Manageable vocabulary size
- Treats different forms of the same word similarly
  - (e.g., “talk”, “talks”, “talked”, “talking”, etc)
- Reduces the impact of the OOV words problem:
  - Segments OOV as subwords and represents the word in terms of these subwords

Byte Pair Encoding (BPE)

- BPE was initially a data compression algorithm:
  - Find the best way to represent data by identifying the common byte pairs
- Used by OpenAI for tokenization when pretraining the GPT model
- Widely used tokenization method among transformer-based LMs
  - GPT, GPT-2, GPT-3, RoBERTa, BART, and DeBERTa, etc.
- Represent the entire text dataset with the least amount of tokens

(Rico Sennrich et al., 2016)

BPE - Training Algorithm

- The training process applies the following steps:
  1. Split the words into characters (after appending </w>)
  2. Create the base vocabulary from the unique characters in the corpus
  3. Compute the frequency of a pair of characters
  4. Merge the most common character pairing
  5. Add this to the list of tokens and recalculate the frequency count for each token
  6. Rinse and repeat **3 to 5** steps until you have reached your **defined token limit** or a **set number of iterations**

(Rico Sennrich et al., 2016)

BPE - Example

- Suppose we are given the following corpus:

Corpus		
newest	lower	low
newest	lower	low
newest	widest	low
newest	widest	low
newest	widest	low

BPE - Example

1. Split the words into characters:
   - Append the **end of the word** (e.g., </w>) symbol to every word in the corpus:

Corpus		
n e w e s t </w>	l o w e r </w>	l o w </w>
n e w e s t </w>	l o w e r </w>	l o w </w>
n e w e s t </w>	w i d e s t </w>	l o w </w>
n e w e s t </w>	w i d e s t </w>	l o w </w>
n e w e s t </w>	w i d e s t </w>	l o w </w>

BPE - Example

2. Create the **base vocabulary** from the unique characters in the corpus:

Frequency	
d	3
e	15
i	3
l	7
n	5
o	7
s	8
t	5
w	15
</w>	15

Vocabulary									
d	e	i	l	n	o	s	t	w	</w>

BPE - Example

Iteration 1: Merge the most common character pairing

1. Compute character pair frequencies:

Frequency	
(d, e): 3	(l, o): 7
(e, r): 2	(n, e): 5
(e, s): 8	(o, w): 7
(e, w): 5	(r, </w>): 2
(i, d): 3	(s, t): 8
(t, </w>): 8	(w, </w>): 5
(w, e): 7	(w, i): 3

BPE - Example

Iteration 1: Merge the most common character pairing

1. Compute character pair frequencies:
2. Merge the most frequent pair

Frequency	
(d, e): 3	(l, o): 7
(e, r): 2	(n, e): 5
(e, s): 8	(o, w): 7
(e, w): 5	(r, </w>): 2
(i, d): 3	(s, t): 8
(t, </w>): 8	(w, </w>): 5
(w, e): 7	(w, i): 3

Corpus		
n e w **es** t </w>	l o w e r </w>	l o w </w>
n e w **es** t </w>	l o w e r </w>	l o w </w>
n e w **es** t </w>	w i d **es** t </w>	l o w </w>
n e w **es** t </w>	w i d **es** t </w>	l o w </w>
n e w **es** t </w>	w i d **es** t </w>	l o w </w>

BPE - Example

Iteration 1: Merge the most common character pairing

1. Compute character pair frequencies
2. Merge the most frequent pair
3. Add them to the vocabulary

Vocabulary									
d	e	i	l	n	o	s	t	w	</w>
es									

BPE - Example

Iteration 2: Merge the most common character pairing

1. Compute character pair frequencies

Frequency	
(d, es): 3	(l, o): 7
(e, r): 2	(n, e): 5
(es, t): 8	(o, w): 7
(e, w): 5	(r, </w>): 2
(i, d): 3	(w, es): 5
(t, </w>): 8	(w, </w>): 5
(w, e): 2	(w, i): 3

BPE - Example

Iteration 2: Merge the most common character pairing

1. Compute character pair frequencies
2. Merge the most frequent pair

Frequency	
(d, es): 3	(l, o): 7
(e, r): 2	(n, e): 5
(es, t): 8	(o, w): 7
(e, w): 5	(r, </w>): 2
(i, d): 3	(w, es): 5
(t, </w>): 8	(w, </w>): 5
(w, e): 2	(w, i): 3

Corpus		
n e w **est** </w>	l o w e r </w>	l o w </w>
n e w **est** </w>	l o w e r </w>	l o w </w>
n e w **est** </w>	w i d **est** </w>	l o w </w>
n e w **est** </w>	w i d **est** </w>	l o w </w>
n e w **est** </w>	w i d **est** </w>	l o w </w>

BPE - Example

Iteration 2: Merge the most common character pairing

1. Compute character pair frequencies
2. Merge the most frequent pair
3. Add them to the vocabulary

Vocabulary									
d	e	i	l	n	o	s	t	w	</w>
es	**est**								

BPE - Example

Iteration 3: Merge the most common character pairing

1. Compute character pair frequencies
2. Merge the most frequent pair

Frequency	
(d, est): 3	(l, o): 7
(e, r): 2	(n, e): 5
(est, </w>): 8	(o, w): 7
(e, w): 5	(r, </w>): 2
(i, d): 3	(w, est): 5
(w, e): 2	(w, </w>): 5
	(w, i): 3

Corpus		
n e w **est</w>**	l o w e r </w>	l o w </w>
n e w **est</w>**	l o w e r </w>	l o w </w>
n e w **est</w>**	w i d **est</w>**	l o w </w>
n e w **est</w>**	w i d **est</w>**	l o w </w>
n e w **est</w>**	w i d **est</w>**	l o w </w>

BPE - Example

Iteration 3: Merge the most common character pairing

1. Compute character pair frequencies
2. Merge the most frequent pair
3. Add them to the vocabulary

Vocabulary									
d	e	i	l	n	o	s	t	w	</w>
es	est	**est</w>**							

BPE - Example

- After 10 iterations the vocabulary is:

Vocabulary									
d	e	i	l	n	o	s	t	w	</w>
es	est	est</w>	lo	low	low</w>	ne	new	newest</w>	

BPE - Training Algorithm

- The training process applies the following steps:
    1. Split the words into characters (after appending </w>)
    2. Create the base vocabulary from the unique characters in the corpus:
    3. Compute the frequency of a pair of characters
    4. Merge the most common character pairing
    5. Add this to the list of tokens and recalculate the frequency count for each token
    6. Rinse and repeat **3 to 5** steps until you have reached your **defined token limit** or a **set number of iterations**

(Rico Sennrich et al., 2016)

What's a shortcoming of BPE?

Always merge greedily based on raw frequency

Other Subword Tokenization

- WordPiece (Yonghui Wu et al., 2016):
  - Merge the pairs based not only on frequency, but also how frequent elements are individually

$$\text{score} = (\text{freq\_of\_pair})/(\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$$

- SentencePiece (Taku Kudo et al., 2018):
  - Train subword models directly from raw sentences
  - Encode everything as Unicode (including spaces)

How many subwords do we need?

Number of Subwords

- The best-performing number depends on:
  - Task & Domain
  - Language
- Having a stopping criteria (as a hyperparameter):
  - Defining the likelihood of a vocabulary with respect to a sequence, and improving that likelihood greedily
- The smaller the dataset, the smaller the subword vocabulary should be

Limitations of Subword Tokenization

- Subwords do not necessarily correspond to morphemes
  - Not optimal for agglutinative languages (e.g., Turkish) and non-concatenative morphology (e.g., Arabic)

كتب	k-t-b	“write” (root form)
كَتَبَ	**kataba**	“he wrote”
كَتَّبَ	**kattaba**	“he made (someone) write”
اِكْتَتَبَ	**iktataba**	“he signed up”

Table 1: Non-concatenative morphology in Arabic. When conjugating, letters are interleaved *within* the root. The root is therefore not separable from its inflection via any contiguous split.

(Jonathan H. Clark et al., 2022)

Limitations of Subword Tokenization

- Pretokenization rules do not work in some languages:
  - Thai, and Chinese do not use spaces between words
  - Hawaiian and Twi use punctuation as consonants

- Struggling with challenging domains:
  - Informal text including typos, spelling variation, transliteration, or emoji

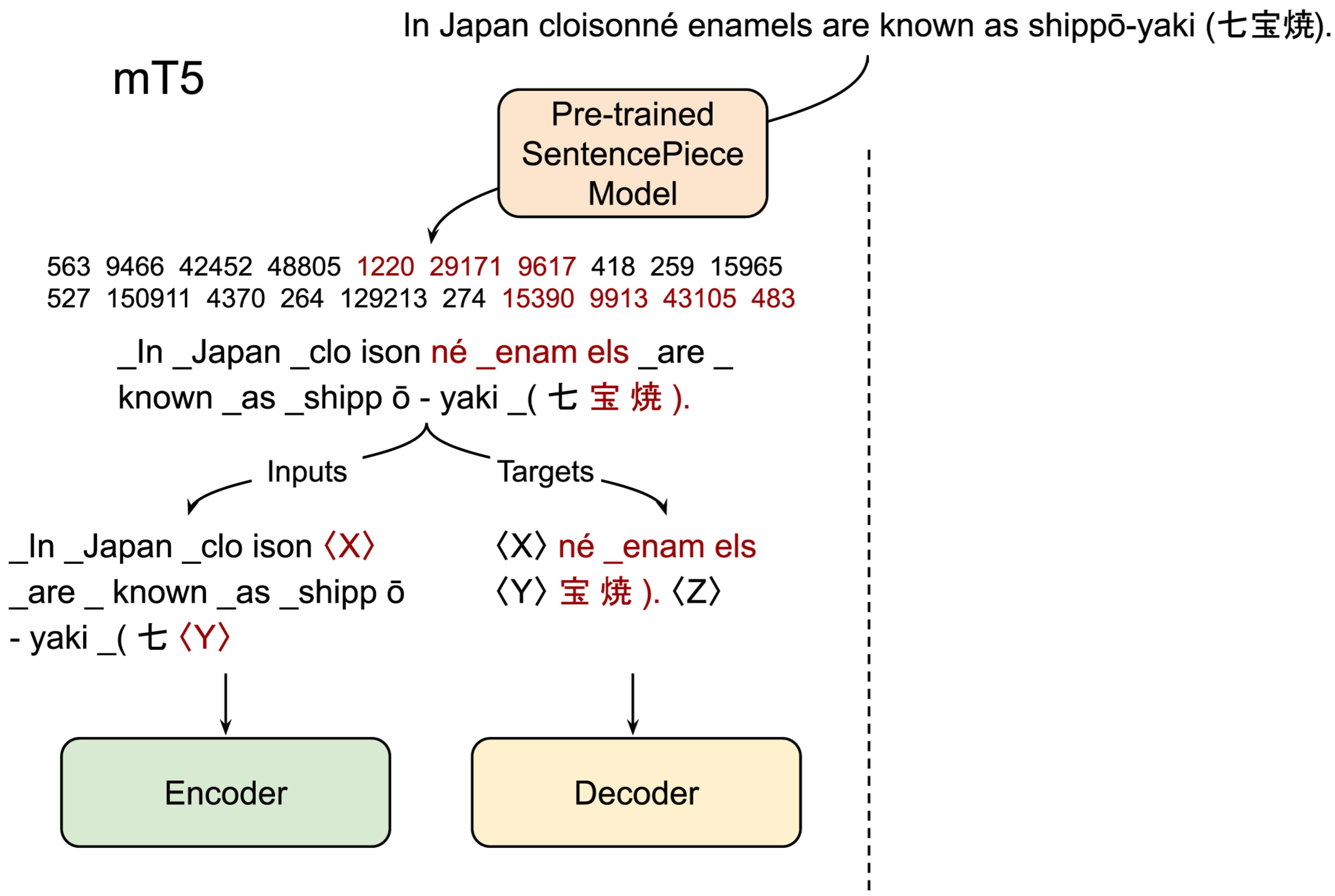
Byte-level Tokenization

- Represent text using the byte sequence resulting from a standard encoding like UTF-8 (e.g., ByT5)
  - Avoiding the huge-vocabulary problem of character-level models
- More robust to noise and out-of-distribution data
- Avoidance of out-of-vocabulary issues

Byte-level Tokenization

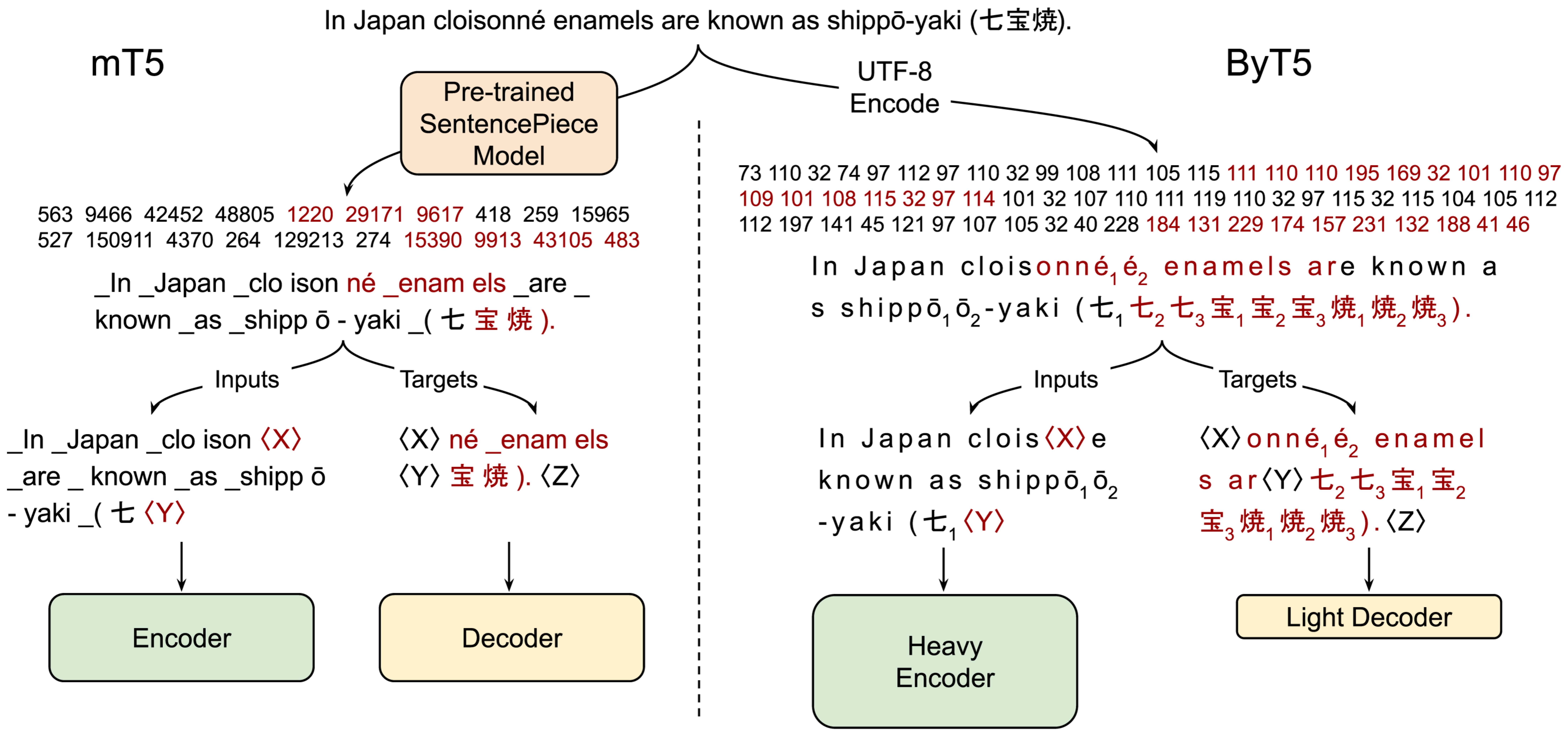
- Represent text using the byte sequence resulting from a standard encoding like UTF-8 (e.g., ByT5)
  - Avoiding the huge-vocabulary problem of character-level models
- More robust to noise and out-of-distribution data
- Avoidance of out-of-vocabulary issues
- **Longer training and inference time due to longer sequences**

ByT5: Tokenizer Free



(Linting Xue et al., 2022)

ByT5: Tokenizer Free



(Linting Xue et al., 2022)

ByT5: Tokenizer Free

- Fewer parameters associated with vocabulary

Size	Params	mT5 Vocab	ByT5 Vocab
Small	300M	85%	0.3%
Base	582M	66%	0.1%
Large	1.23B	42%	0.06%
XL	3.74B	27%	0.04%
XXL	12.9B	16%	0.02%

- Improvement in tasks with noisy data

- Dealing with increased sequence length:
    - Training with shorter sequences (max 1024 bytes)

(Linting Xue et al., 2022)

References

- Mielke, Sabrina J., et al. "Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP." arXiv preprint arXiv:2112.10508 (2021).

- Xue, Linting, et al. "Byt5: Towards a token-free future with pre-trained byte-to-byte models." Transactions of the Association for Computational Linguistics 10 (2022): 291-306.

- Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." arXiv preprint arXiv:1508.07909 (2015).