Tokenization

CS685 Fall 2021

Advanced Natural Language Processing

Mohit Iyyer

College of Information and Computer Sciences

University of Massachusetts Amherst

Stuff from last time

- Practical fine-tuning tips?
- Midterm date? Released Nov 9, you will have 48 hours to complete it
- Will release previous years' midterms on Piazza soon

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. 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!

- 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

Other subword encoding schemes

- WordPiece (Schuster et al., ICASSP 2012): merge by likelihood as measured by language model, not by frequency
- SentencePiece (Kudo et al., 2018): can do subword tokenization without pretokenization *(good for languages that don't always separate words w/ spaces)*, although pretokenization usually improves performance

Limitations of subwords

- Hard to apply to languages with agglutinative (e.g., Turkish) or non-concatenative (e.g., Arabic) morphology

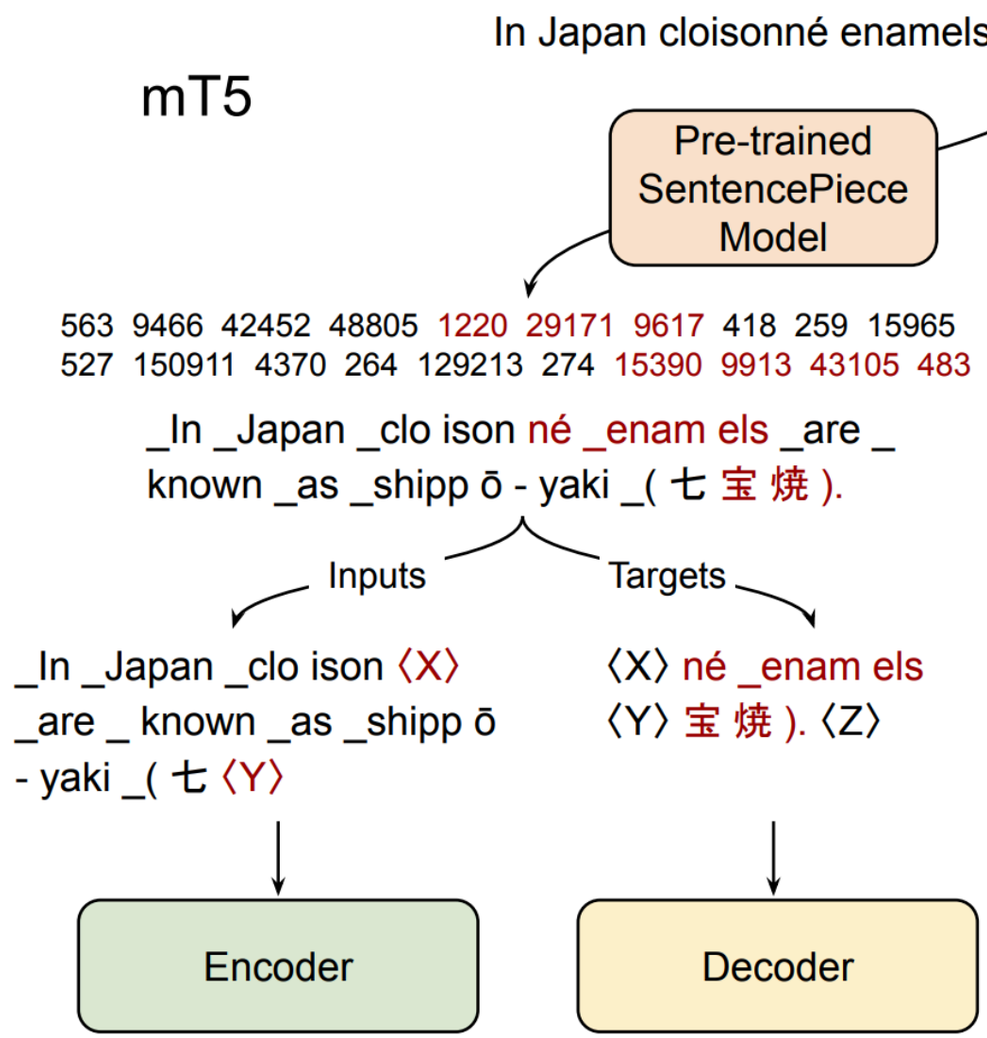
- Pretokenization rules don’t work on some languages (Thai, Chinese don’t use spaces between words; Hawaiian uses punctuation as consonants)

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.

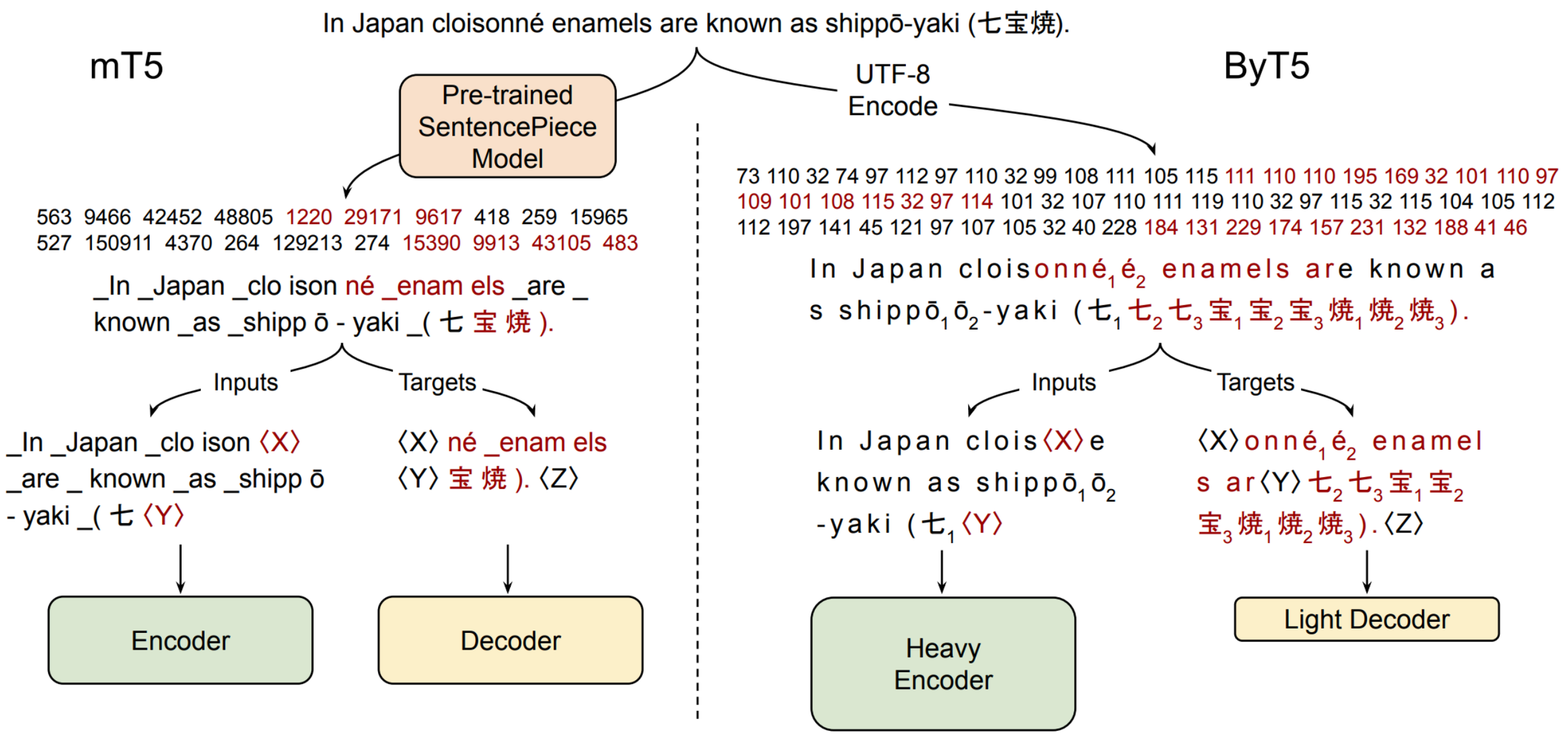
Clark et al., 2021, “CANINE”

ByT5: tokenizer free!



Xue et al., 2021, "ByT5"

ByT5: tokenizer free!



Xue et al., 2021, "ByT5"

Way fewer params associated with vocabulary!

		mT5				ByT5				
Size	Params	Vocab	d_{model}	d_{ff}	# Enc/Dec	Vocab	d_{model}	d_{ff}	# Enc	# Dec
Small	300M	85%	512	1024	8	0.3%	1472	3584	12	4
Base	582M	66%	768	2048	12	0.1%	1536	3968	18	6
Large	1.23B	42%	1024	2816	24	0.06%	1536	3840	36	12
XL	3.74B	27%	2048	5120	24	0.04%	2560	6720	36	12
XXL	12.9B	16%	4096	10240	24	0.02%	4672	12352	36	12

Impressive gains on tasks w/ noisy data

		Learnable Noise		Unseen Noise
	Model	XNLI (accuracy)	TyDiQA-GoldP (F1)	XNLI (accuracy)
Clean	mT5	81.1	85.3	81.1
	ByT5	79.7	87.7	79.7
Drop	mT5	−10.2	−19.9	−18.3
	ByT5	**−8.2**	**−18.4**	**−11.4**
Add/Drop/Mutate	mT5	−9.2	−28.5	−11.4
	ByT5	**−8.0**	**−24.3**	**−10.9**
Repetitions	mT5	−8.5	−11.0	−12.3
	ByT5	**−4.1**	**−3.1**	**−5.9**
Antspeak	mT5	−32.0	−17.5	−34.4
	ByT5	**−8.7**	**−4.3**	**−24.4**
Uppercase	mT5	−7.0	−7.6	−8.1
	ByT5	**−1.5**	**−1.0**	**−1.7**
Random Case	mT5	−25.7	−13.9	−19.2
	ByT5	**−1.5**	**−1.2**	**−5.9**

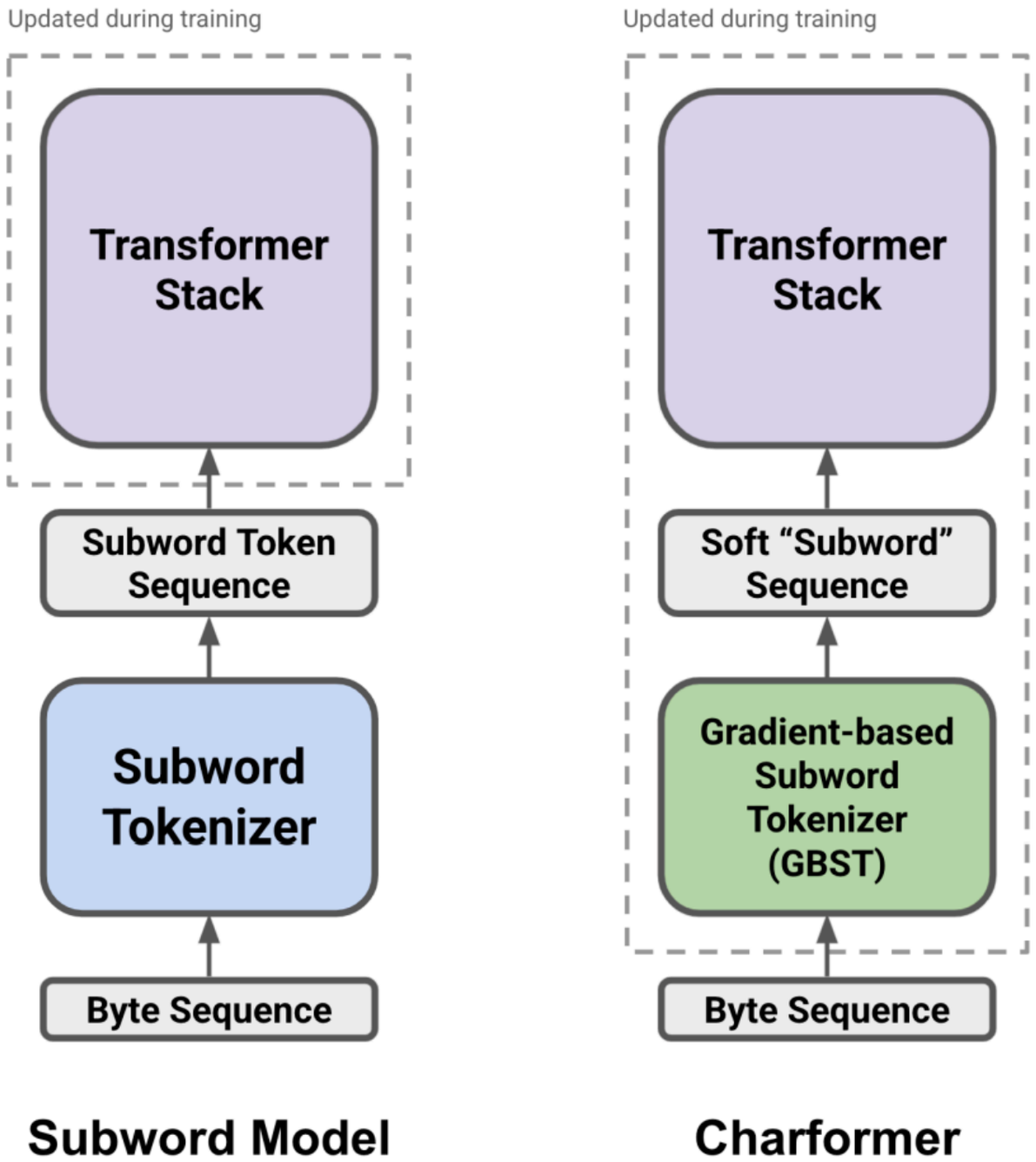
How to deal w/ increased sequence length?

- ByT5: just train with shorter sequences (mT5 is trained on max length 1024 subword tokens, ByT5 trained on max 1024 bytes)

- At test-time, ByT5 can be 7X slower than mT5 to generate sentences

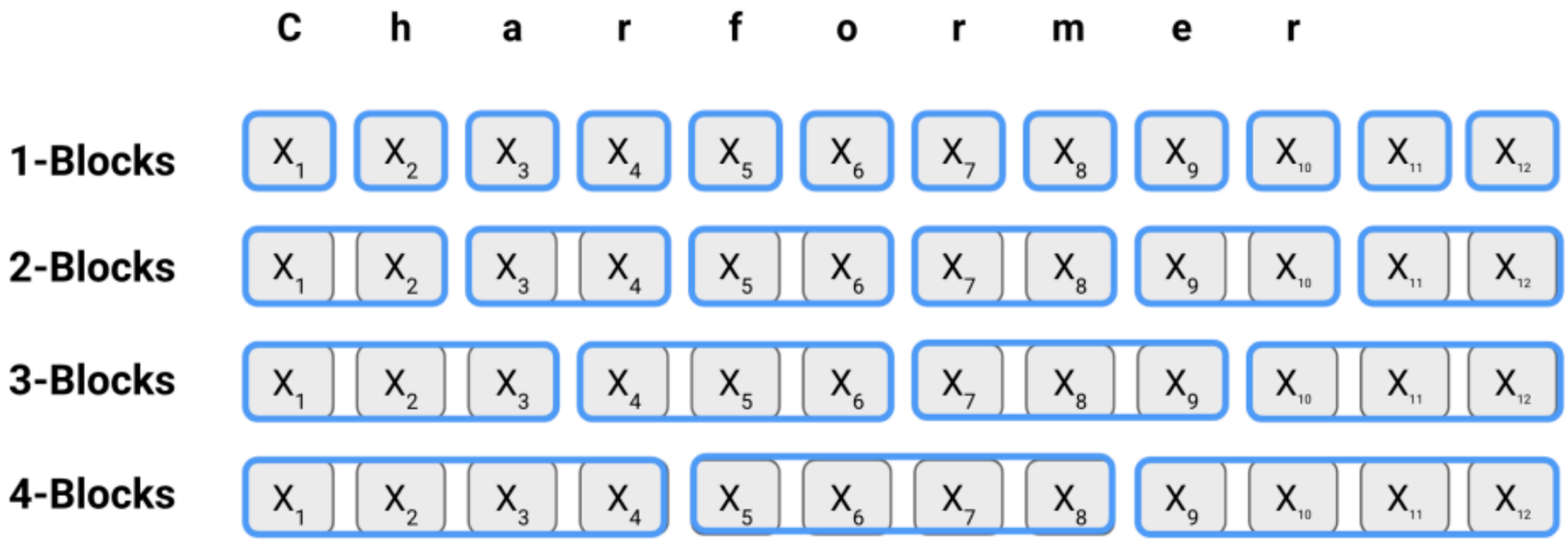
- Later this semester: use more efficient Transformer architectures

Learnable tokenization



Tay et al., 2021, “Charformer”

Consider multiple segmentations



Then, for each character, score all blocks to which that character belongs