Tokenization and Tokenizer-Free LMs

CSE 5539: Advanced Topics in Natural Language Processing

https://shocheen.github.io/courses/advanced-nlp-fall-2024

Logistics

- Mid-term report: due tonight
- Quiz also due tonight – just 3 multiple choice questions.


Cody Blakeney
@code_star
Do you guys ever think about tokenizers?


Prithviraj (Raj) Ammanabrolu @rajammanabrolu · Sep 26, 2023
If there's one thing standing in the way of AGI, it's tokenizers


Luca Soldaini @soldni · Aug 21
0
DAYS WITHOUT
TOKENIZATION
ACCIDENTS
0
imgflip.com

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**.

[`Let's build the GPT Tokenizer' by Andrej Karpathy]

Agenda

I. Prior and current tokenization practices and their issues
    I. Word-level tokenization
    II. BPE, WordPiece, Unigram
II. Tokenization-free language models
    I. Byte-level models
    II. Fixed-length patching: Charformer, Canine et al.
    III. Dynamic Patching: Learnable tokenization
III. A brief look at “tokenization” in other modalities.

What's tokenization

Recall: A language model is a probability distribution over a sequence of "tokens"; each token is from a "vocabulary".

What is a token: "basic unit that need not be decomposed for further processing" – traditionally defined as a word
[Webster and Kit, 1992]

Tokenization: Splitting or segmenting a "string" of text into a sequence of *tokens*

"I love watching the television"

$$\bar{X} = \langle \text{I, love, watching, the, television} \rangle$$

Why do we need to tokenize?

- Tokenization is not a typical preprocessing step in most machine learning domains
- Neural networks work with real-valued numbers. Most machine learning deals with continuous data but text is discrete
- Text needs to be converted to a form that a model can consume/generate.

Set of all tokens form a *vocabulary*

- Given a tokenization algorithm
  - Tokenize a corpus of text
  - Collect all unique tokens (aka types) → vocabulary

- The role of vocabulary in a language model
  - The vocabulary and its size is part of the model architecture
  - It defines the size of the input embedding table and final output layer
    - large vocabulary = more parameters

A simple tokenizer: Split by whitespace?

"I love watching the television"

$$\bar{x} = \langle \text{I, love, watching, the, television} \rangle$$

- Tokenization is not simple, and tokenizers require many specialized rules
- Such as, what will we do with the following strings:
  - "amazing!", "state-of-the-art", "unthinkable", "prize-winning", "aren't", "O'Neill"
  - Some languages don't even use spaces to mark word boundaries!

私は日本語を勉強しています

I am studying Japanese.

Çekoslovakyalılaştıramadıklarımızdanmışsınız

You are one of those whom we could not turn into a Czechoslovakian.

What about a word level tokenizer?

- Define each token as a word or a punctuation:
  - What is a word: smallest unit of language that carries meaning and can stand alone or combine with other units to create more complex meanings

- How to split into words?
  - For languages like English:
    - Could be simple regexes: split on all spaces and punctuation
      - What about “The value of pi = 3.14”, “the IP address is 0.0.0.0”
      - What about “He got cold feet” → should cold feet be one or two words?
  - For languages like Chinese:
    - 我爱自然语言处理 (I love Natural Language Processing)
    - Need specialized tools (e.g. jieba)

Some related terminology

- A **morpheme** is the smallest meaning-bearing unit of a language
  - “unlikeliest” has the morphemes {un-, likely, -est}
- **Morphology** is the study of the way words are built up from morphemes
- **Word forms** are the variations of a word that express different grammatical categories
  - **Tense** (when something happened; past, present, future)
  - **Case** (inflecting nouns/pronoun and their modifiers to express their relationship with other words; English has largely lost its inflected case system)
  - **Number** (singular/plural)
  - **Gender** (masculine, feminine, neuter; not extensively used in English)

and thus help convey the specific meaning and function of the word in a sentence

Word level tokenizer – How to define a vocabulary?

- Given a tokenization algorithm
  - Tokenize a corpus of text
  - Collect all unique tokens → vocabulary
- For a English corpus, a corpus like OpenWebText can have 1M+ unique words
  - Vocabulary becomes too large
- A popular solution: Cut this list to include only K tokens.
  - How to chose K – based on frequency
  - What to do with the rest? – replace with an UNK “unknown” token
  - Word level tokenizers lead to “closed” vocabulary models.

Handling Unknown Words

- What happens when we encounter a word that we have never seen in our training data?
  - Not much we can do
  - Except assigning to it a special <UNK> token
  - Why this is bad?

Limitations of <UNK>

- Generally, we lose most of the information the word conveys. UNKs don't give features for novel words that are useful anchors of meaning
  - What if you want to generate a word which is not in the vocabulary? Imagine ChatGPT generating UNK.
- Especially hurts in [productive] languages with many 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 ").*

Word Level Tokenization

Other Limitations

- Word-level tokenization treats different forms of the same root as completely separate (e.g., “open”, “opened”, “opens”, “opening”, etc)
- This means separate features or embeddings.
  - Why is this a problem?

But deciding what counts as a word in Chinese is complex. For example, consider the following sentence:

(2.4) 姚明进入总决赛
“Yao Ming reaches the finals”

As Chen et al. (2017) point out, this could be treated as 3 words (‘Chinese Treebank’ segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words (‘Peking University’ segmentation):

(2.6) 姚 明 进入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

In fact, for most Chinese NLP tasks it turns out to work better to take characters rather than words as input, since characters are at a reasonable semantic level for most applications, and since most word standards, by contrast, result in a huge vocabulary with large numbers of very rare words (Li et al., 2019).

A simpler tokenizer: Character-level

I love watching the television

<I _ l o v e _ w a t c h i n g _ t h e _ t e l e v i s i o n>

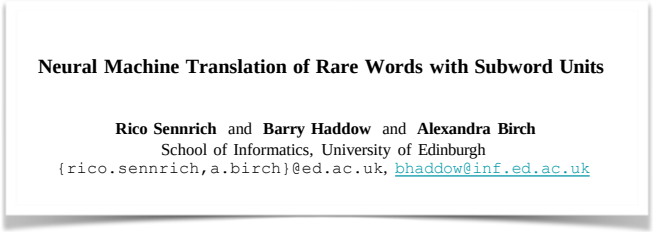
Issues

- Make sequences very long
- Word-level models provide an inductive bias to models. BUT character-level models must learn to compose characters into words.
    - Need deeper models

[Jurafsky and Martin Sec 2.4 p17]

Current standard: *Subword* Tokenization

- "Word"-level: issues with unknown words and information sharing, and gets complex fast
  - Also, fits poorly to some languages
- Character-level: long sequences, the model needs to do a lot of heavy lifting in representing that is encoded in plain-sight
- **Let's find a middle ground!**
- Subword tokenization first developed for machine translations
  - Based on byte pair encoding (Gage, 1994)
- Now, used everywhere



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

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.

A redefinition of the notion of tokenization

Due to:

- Scientific results: The impact of sub-word segmentation on machine translation performance in 2016
- Technical requirements: A fixed-size vocabulary for neural language models & a reasonable vocabulary size

...in current NLP, the notion of *token* and *tokenization* changed

"Tokenization" is now the task of segmenting a sentence into non-typographically (& non-linguistically) motivated units, which are often smaller than classical tokens, and therefore often called **sub-words**

Typographic units (the "old" tokens) are now often called **"pre-tokens",** and what used to be called "tokenization" is therefore called nowadays **"pre-tokenization"**

- https://github.com/huggingface/tokenizers/tree/main/tokenizers/src/pre_tokenizers

[Mielke et al., 2021]

Byte-Pair-Encoding (BPE)

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

Main idea: Use our data to automatically tell us what the tokens should be

Token learner:
Raw train corpus ⇒ Vocabulary (a set of tokens)

Token segmenter:
Raw sentences ⇒ Tokens in the vocabulary

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – Token learner

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

Raw train corpus ⇒ Vocabulary (a set of tokens)

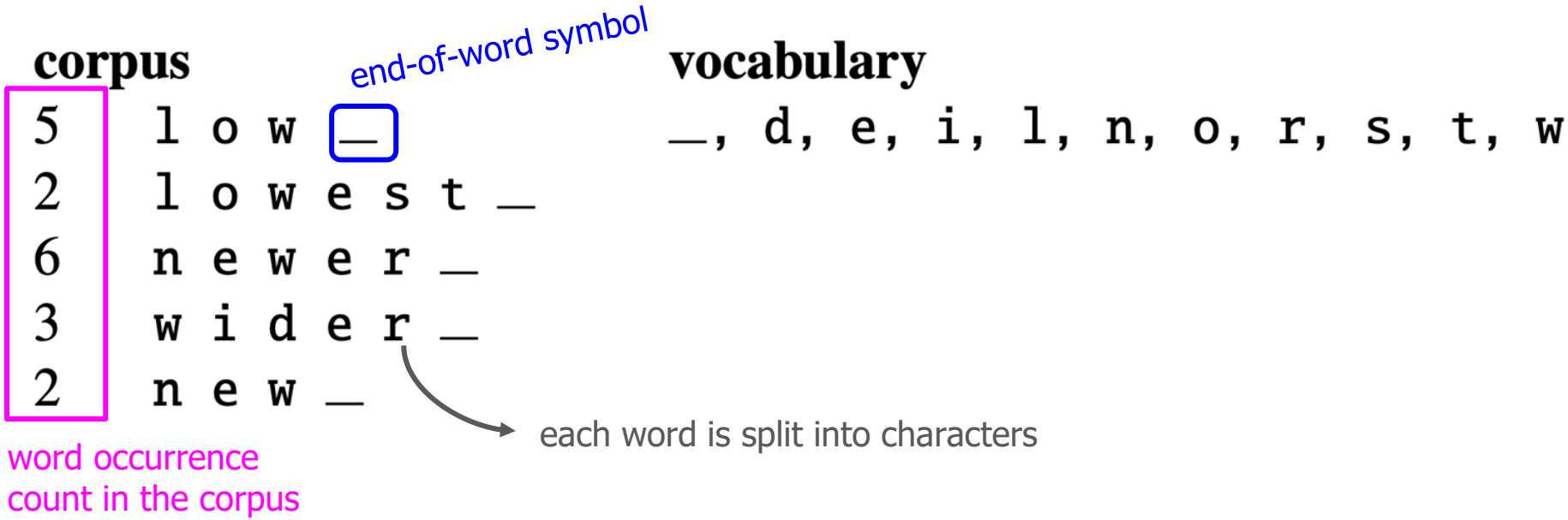
- Pre-tokenize the corpus in words & append a special end-of-word symbol _ to each word
- Initialize vocabulary with the set of all individual characters
- Choose 2 tokens that are most frequently adjacent (“A”, “B”)
  - Respect word boundaries: Run the algorithm inside words
- Add a new merged symbol (“AB”) to the vocabulary
- Change the occurrence of the 2 selected tokens with the new merged token in the corpus
- Continues doing this until k merges are done

All k new symbols and initial characters are the final vocabulary

What’s k? Open research question, see Mielke et al., 2021 Sec 6.6; 30K is seen frequently

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – Token learner *Example*



[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `e r` [a total of 9 occurrences]
- Merge these symbols, treating `er` as one symbol, & add the new symbol to the vocabulary

corpus

```
5   l o w _
2   l o w e s t _
6   n e w er _
3   w i d er _
2   n e w _
```

vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er
```

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `er _`
- Merge these symbols, treating `er_` as one symbol, & add the new symbol to the vocabulary

corpus

```
5   l o w _
2   l o w e s t _
6   n e w er_
3   w i d er_
2   n e w _
```

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair n e
- Merge these symbols, treating ne as one symbol, & add the new symbol to the vocabulary

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Final vocabulary

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token segmenter

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

- The token segmenter is used to tokenize a test sentence
  - Just runs on the test data the merges we've learned from the training data, greedily, in the order we learned them
- First, we segment each test sentence word into characters
- Then, we apply the first merge rule
  - E.g., replace every instance of `e r` in the test corpus with `er`
- Then the second merge rule
  - E.g., replace every instance of `er _` in the test corpus with `er_`
- And so on

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token segmenter

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

- Test example: slow_ → s l o w_ → s lo w_ -> s low_
- Test example: now → n o w

BPE can tokenize a word never seen at training time.

Leads to an open vocabulary model*

Can often learn morphological segmentations

Deescalation → De escalat ion

[Example from Jurafsky & Martin (2023), pages 18–19]

A variant of BPE: WordPiece

used in BERT and some follow ups

- Training algorithm: Same as BPE
- Tokenization algorithm: greedily pick the longest prefix that exists in the vocabulary (and repeat)
    - Slow_ → s low_ [stop]

BPE/Wordpiece summary

- Start with a character vocabulary
- Merge frequent bigrams
- Repeat until a desired vocab size/merge size is reached.

Unigram LM Tokenizer

1. Start with a **large base vocabulary**, **remove tokens** until a desired size is reached.

1. **How to construct a base vocabulary**: all substrings of pre-tokenized words OR start with a large BPE vocabulary.

1. **How to remove tokens**:
    a. Compute **the unigram LM loss** over the corpus (more details later)
    b. Removing tokens increases this loss.
    c. Select and remove tokens that increase it the least.
    d. Repeat

Base Vocabulary

- The corpus:

  ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- Initial Vocabulary (all strict substrings)

  ("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20) ("p", 17) ("pu", 17) ("n", 16)

  ("un", 16) ("b", 4) ("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5)

Unigram LM loss

- Unigram LM loss = negative log probability of the corpus.

- Probability of a corpus = product of marginal probability of individual words

  p(pug hugs bugs) = p(pug) x p(hugs) x p (bugs)

Probability of a word

- product of marginal probability of its subwords (based on frequency)

  p(pug) = p(“p”) x p(“u”) x p(“g”)

  Or, p(pug) = p(“pu”) x p(“g”)

  Or, p(pug) = p(“p”) x p(“ug”)

  Choose highest of all possible splits (why?)

- How to this efficiently: Dynamic programming (Viterbi algorithm)

Unigram Tokenization Algorithm

1. Start with a base vocabulary

1. Compute the unigram loss, L, over the corpus

VERY SLOW!

1. For every token, w, in the vocabulary
   a. Remove w from the vocabulary and recompute the loss, L'(w)
   b. Define score(w) = L'(w) - L

1. Compute w* = min_w score(w).
   a. Remove w* from the vocabulary.
   b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

1. Compute the unigram loss, L

1. For every token, w, in the vocabulary
   a. Remove w from the vocabulary and recompute the loss, L'(w)
   b. Define score(w) = L'(w) - L

1. Compute **W** = x% of tokens with the lowest score.
   a. Remove **W** from the vocabulary.
   b. Go to step 2.

How to tokenize once the vocabulary is decided

- Viterbi again – tokenization which maximizes the unigram probability of the word
  - (or find top *k* tokenizations)

- “Unhug” (For each position, the subwords with the best scores ending in that position:)

  Character 0 (u): "u" (score 0.171429)

  Character 1 (n): "un" (score 0.076191)

  Character 2 (h): "un" "h" (score 0.005442)

  Character 3 (u): "un" "hu" (score 0.005442)

  Character 4 (g): "un" "hug" (score 0.005442) **[final tokenization]**

Unigram vs BPE

1. Why unigram over BPE and WordPiece?
    a. Unigram finds optimal coding length of a sequence (according to Shannon's entropy).
    b. Unigram allows sampling multiple tokenizations for every word – subword regularization – robustness

1. Why is Unigram tokenization not more popular?
    a. In many papers simply referred to as SentencePiece, which is actually not a tokenizer but a library wrapping several tokenizers
    b. Does the actual subword tokenizer matter as we scale up models?

Subword methods are not TRULY open-vocabulary

- What if you encounter a character not seen at training time?

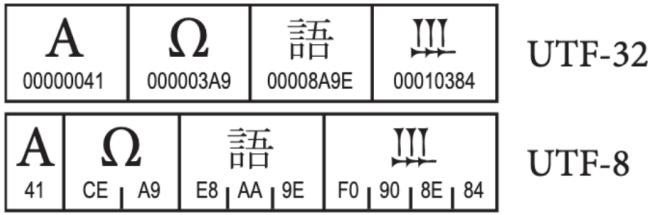
- For example, a BPE model trained on only English encounters a Chinese character at test time --- it gets assigned UNK

- How to solve this issue?
  - Train on a mix of all characters? Accounting for all languages, they are millions of characters – vocabulary size would be too large

Solution: Byte-level subword Models (BBPE)

- Every written language is represented in Unicode

Unicode is a text encoding standard designed to support the use of text in all of the world's writing systems that can be digitized. It has multiple versions like UTF-8, UTF-16, UTF-32. UTF-8 is the most common one.



- Each character is a sequence of “bytes”.
  - Each byte is 8-bit. Total 256 unique byte values
  - Each character requires between 1 to 4 bytes.

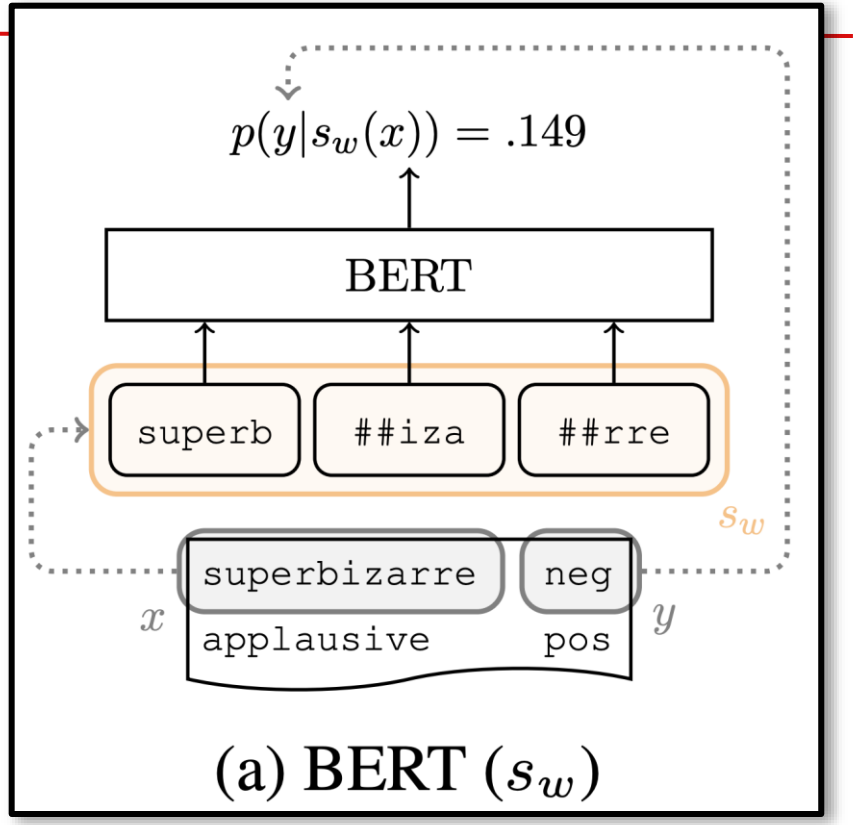
Solution: Treat a corpus as a sequence of byte. Start the vocabulary with all bytes (256) and train a BPE model. TRULY Open Vocabulary. Works for all characters!*

Subword Models -- Summary

- Split text in tokens learned statistically from the training corpus
- Makes the model open vocabulary
- Subword methods prove to be an effective method of “compressing text”

Issues with subword models

BERT thinks the sentiment of "superbizarre" is positive because its tokenization contains the token "superb"



[Hofmann et al., 2021]

Non-concatenative Languages

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.

Subword Tokenization and “noise”

- He fell and broke his **coccix** (vs coccyx)
- Neighbor = 1 token, neighbour = two tokens
- John Smith = 2 tokens, Srinivas Ramanujam = ??

Subword Tokenization and “numbers”

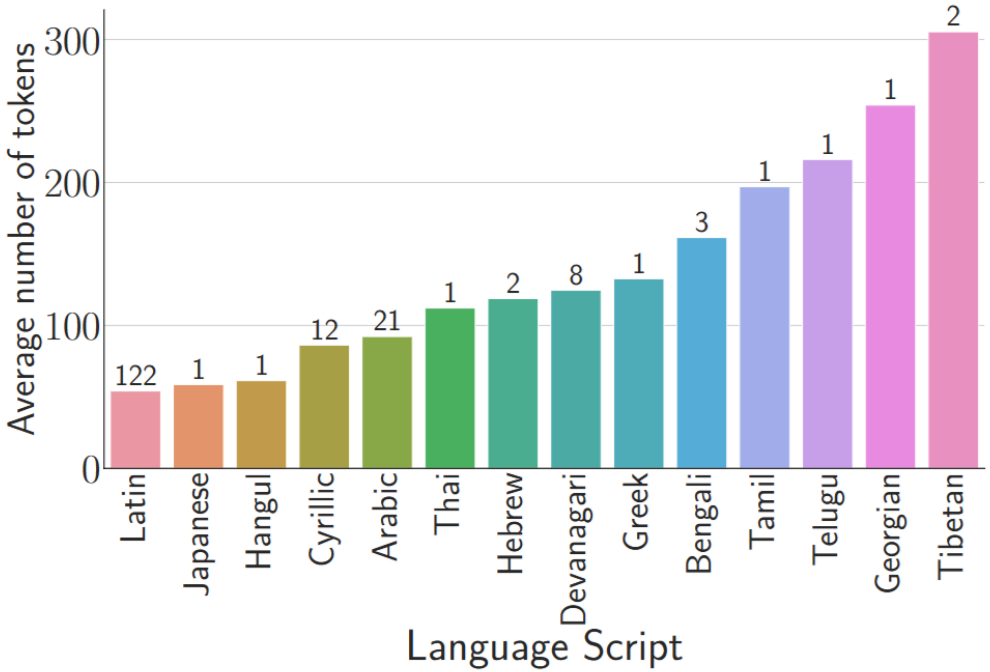
- Why are LMs bad at basic arithmetic?
  - 3.14 is tokenized as 3.14
  - 3.15 is tokenized as 3 . 15

Natural phenomena like diacritics or a little easily human-readable noise lead to unexpected BPE sequences and catastrophic translation failure

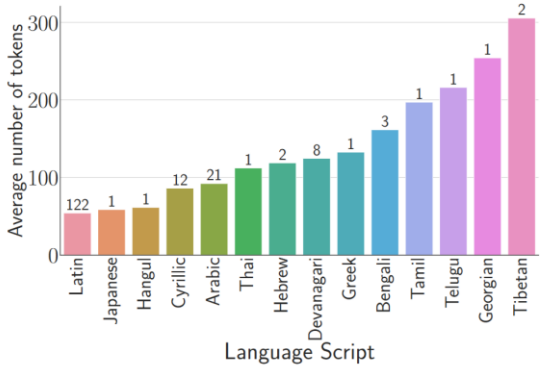
Arabic–English	
src	أنا كندية، وأنا أصغر أخواني السبعة
diacritics 1.0	أَنا كَنَدِيَّةٌ ، وَأَنا أَصْغَرِ إِخْوانِي السَبْعَةِ
ref	I'm Canadian, and I'm the youngest of seven kids.
in_{vis}	
out_{vis}	I'm a Canadian, and I'm the youngest of my seven sisters.
COMET	0.764
in_{text}	▁أَ ان ▁ك ن د ي ة ▁, ▁و ▁أ ان ▁أ ص غ ر ▁إ خ وا ن ي ▁سلا ب ع ة ▁.
out_{text}	We grew up as a teacher, and we gave me a hug.
COMET	-1.387

[Salesky et al., 2021]

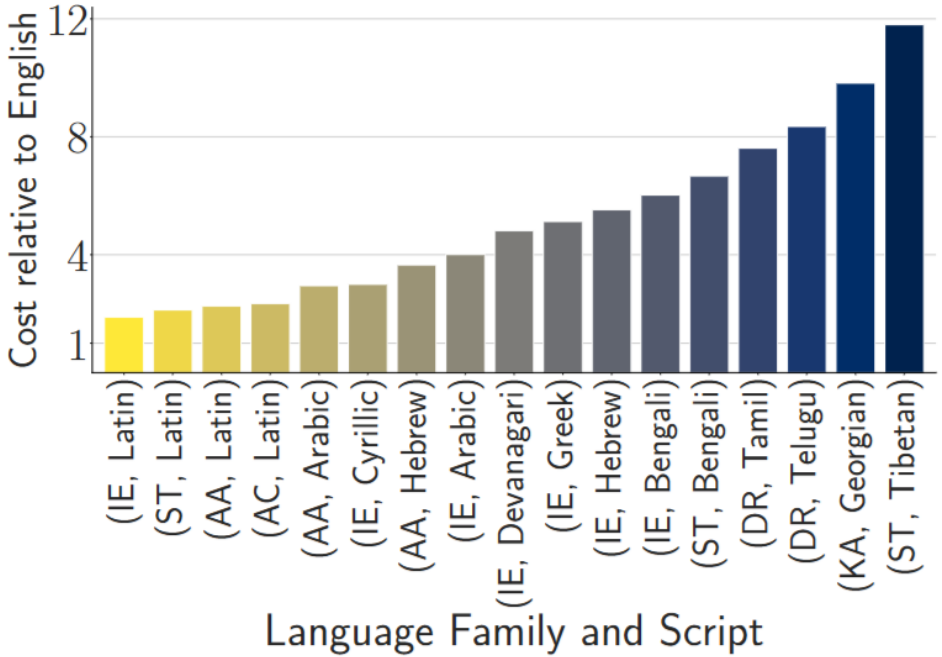
Sequence Lengths, Costs, and Performance



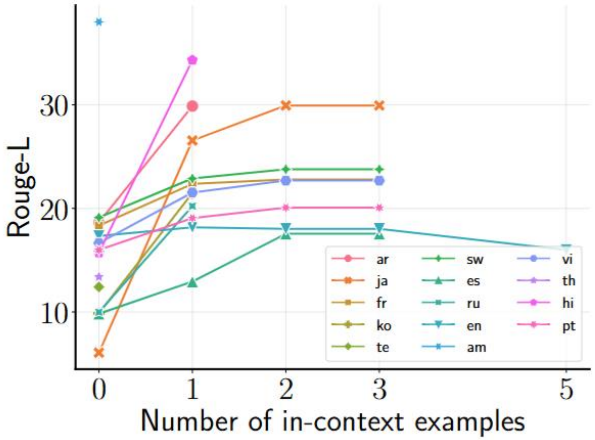
Sequence Lengths, Costs, and Performance



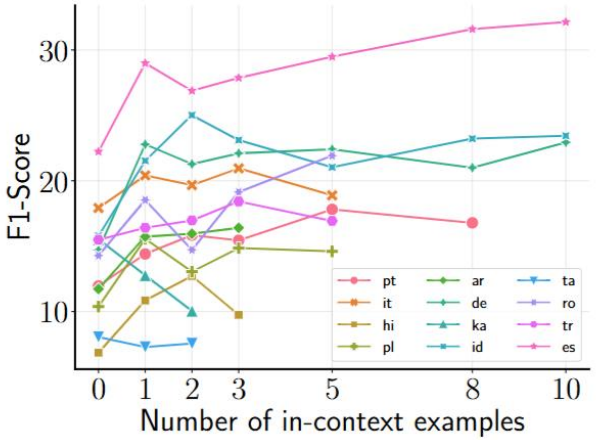
LLM APIs charge per token



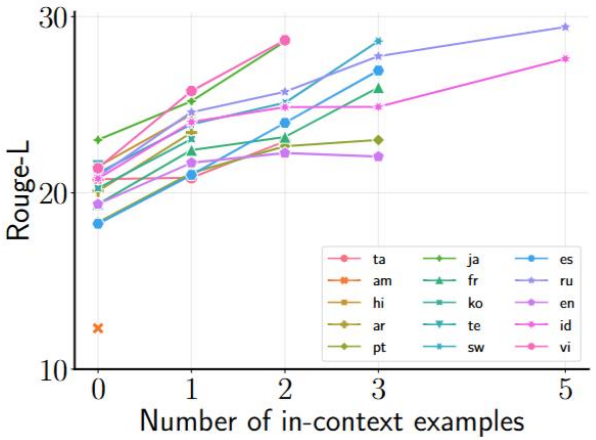
Sequence Lengths, Costs, and Performance



(a) XLSUM



(b) XFACT



(c) CROSS-SUM

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**.

[`Let's build the GPT Tokenizer' by Andrej Karpathy]

Recent tweaks to subword tokenizers

- Tokenize each digit separately (i.e. no merge on digits)

- Add special tokens to deal with everything else:
  - E.g. special tokens for keywords from programming languages

- Train the tokenizer on a more balanced dataset.
  - Apply tricks like alpha-sampling – up sample lower resource languages and scripts to up their frequency.
  - Does not solve all issues but helps.

Tokenizer-free Models

Goal: Eliminate the need to have a **separately** trained tokenizer.

Character/Byte-level Language Models

- Return of the character: Directly model characters or bytes

- How to deal with the inefficiencies?
  - Modify the model architecture

Why character/byte level models

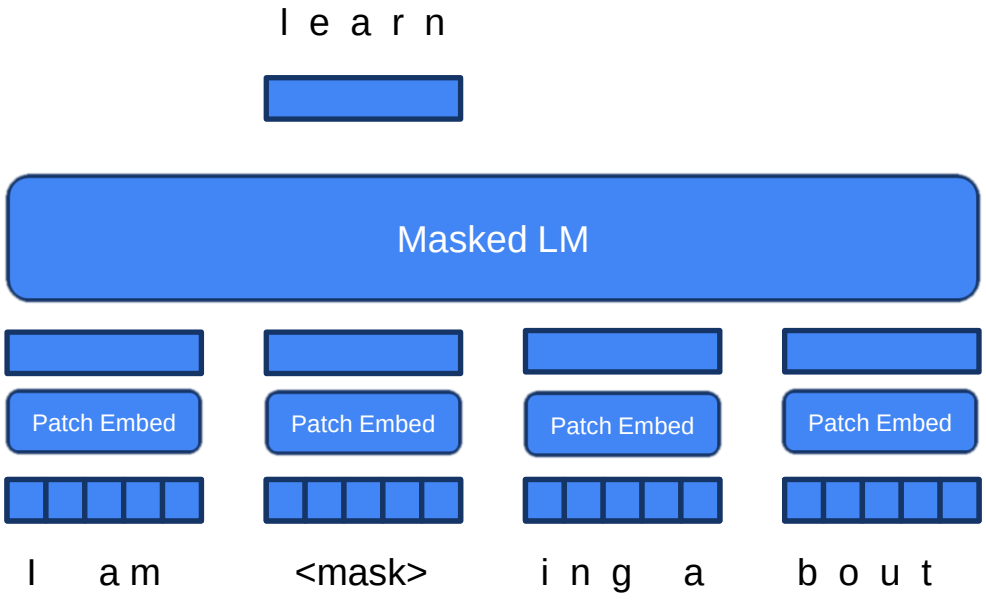
- No linguistic inductive bias, let the model figure out how it needs to compose bytes to do the task at hand
  - Especially useful for languages for which we don't have enough data or pretokenizers

- Allows models to deal with noisy inputs (spelling errors, language variation)

Tokenizer-Free Approaches: Efficiency

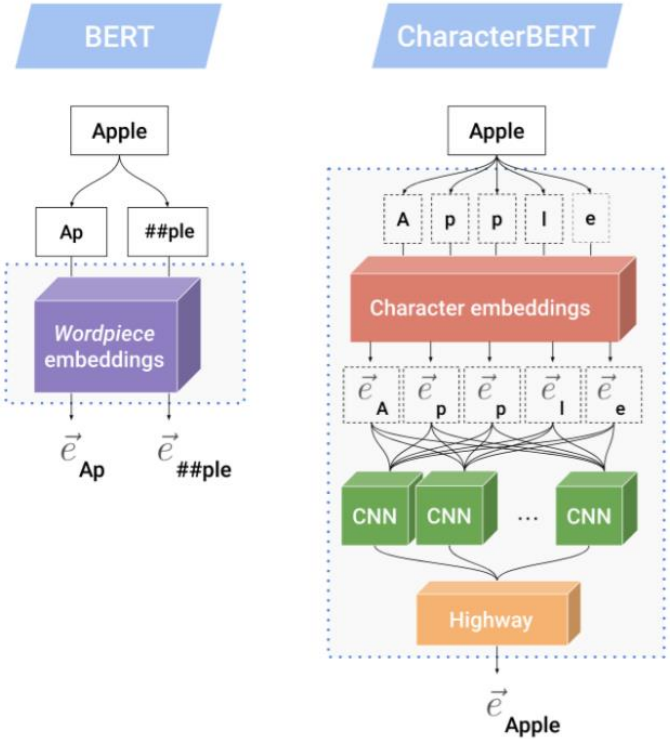
- Encoder Only: CharacterBERT
  Use a CNN to encode characters to word-level representations
- Encoder Only: Canine (Clark et al. 2022)
  Downsample representation of local one-layer attention.
- Encoder-decoder: ByT5 (Xue, et al. 2022)
  Uses shallow decoder, predicts the masked patches of ~20 bytes (for comparison 3 tokens in mT5).
- Encoder-decoder: CHARFORMER (Tay, Tran, et al. 2022)
  Uses convolution to obtain representation of variable length blocks of characters.
- Decoder-only: MEGABYTE (Yu et al. 2023)
  Autoregressive models trained on byte sequences (grouped into 4-element patches). Text + Images.
- Decoder-only: “Dynamic Token Pooling” (Nawrot et al., 2023)
  Local representation based on pooling variable-length of tokens. Auxiliary loss used for segmentation.

- Basic idea: use a shallow network to compose characters/bytes into longer representations. Apply masked language modelling loss.



Encoder Only Models

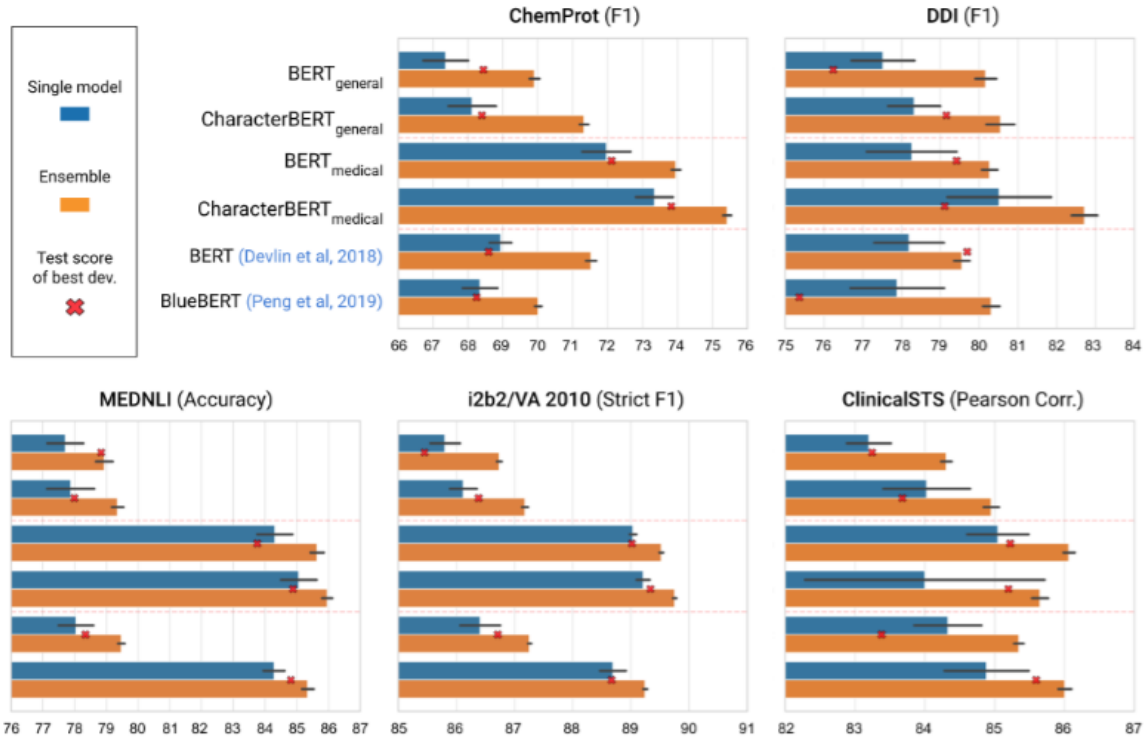
- CharacterBERT



Requires word tokenization, technically a word-level model

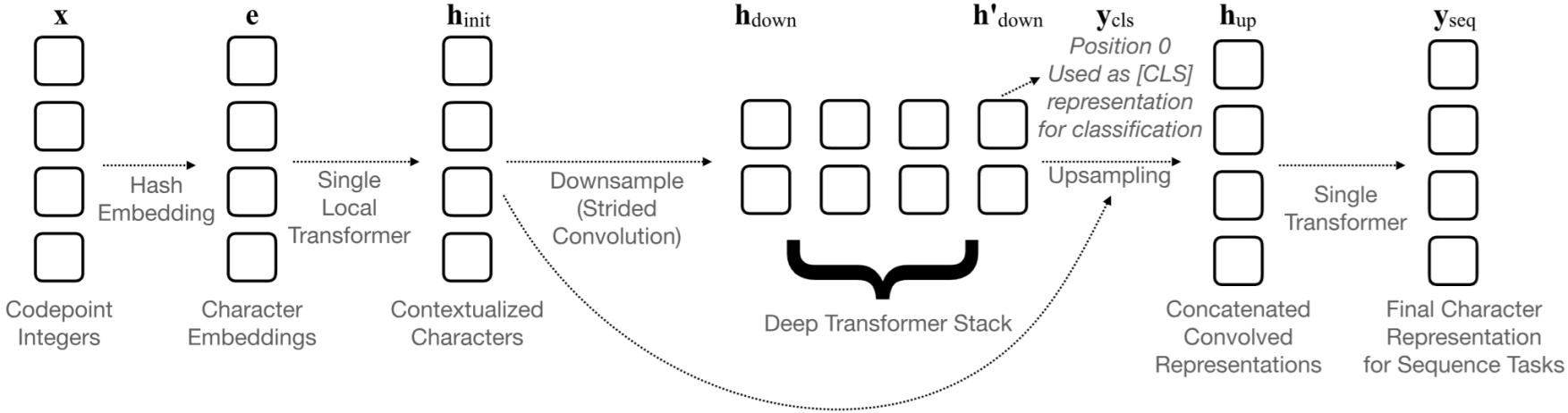
Loss is masked language modelling – uses top 100K words to compute loss

CharacterBERT



Encoder Only Models

- CANINE



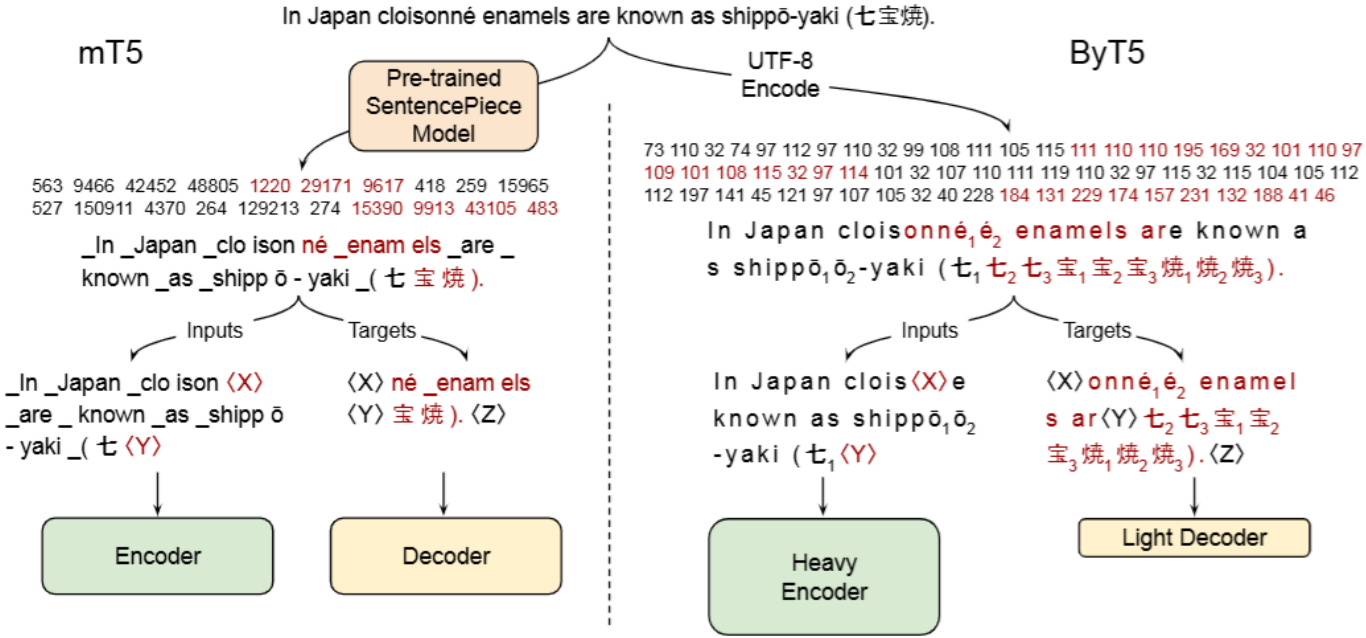
No need for any proprocessing
Loss is still masked language modelling but computed over bytes

Implicitly tokenizes the input string to equal length tokens.

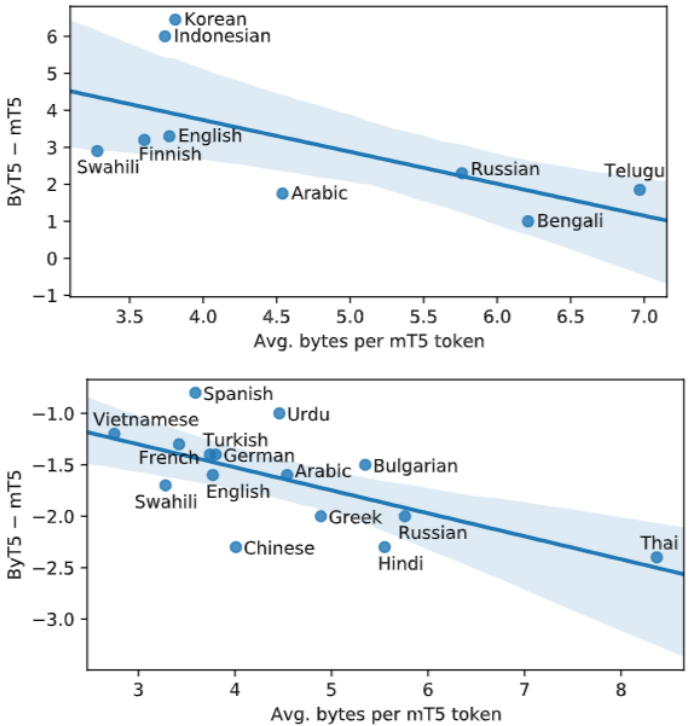
CANINE

Model	Input	MLM	r	Length	Examples / sec	Params	TyDiQA SelectP	TyDiQA MinSpan
mBert (public)	Subwords	Subwords	–	512	–	179M	63.1	50.5
mBert (ours)	Subwords	Subwords	–	512	9000	179M	63.2	51.3
	Chars	Single Chars	1	2048	925	127M	59.5 (–3.7)	43.7 (–7.5)
	Chars	Subwords	1	2048	900	127M	63.8 (+0.6)	50.2 (–1.0)
Canine-S	Chars	Subwords	4	2048	6400	127M	66.0 (+2.8)	52.5 (+1.2)
Canine-C	Chars	Autoreg. Chars	4	2048	6050	127M	65.7 (+2.5)	53.0 (+1.7)
Canine-C + n-grams	Chars	Autoreg. Chars	4	2048	5600	167M	**68.1** (+4.9)	**57.0** (+5.7)

Encoder-decoder model: ByT5



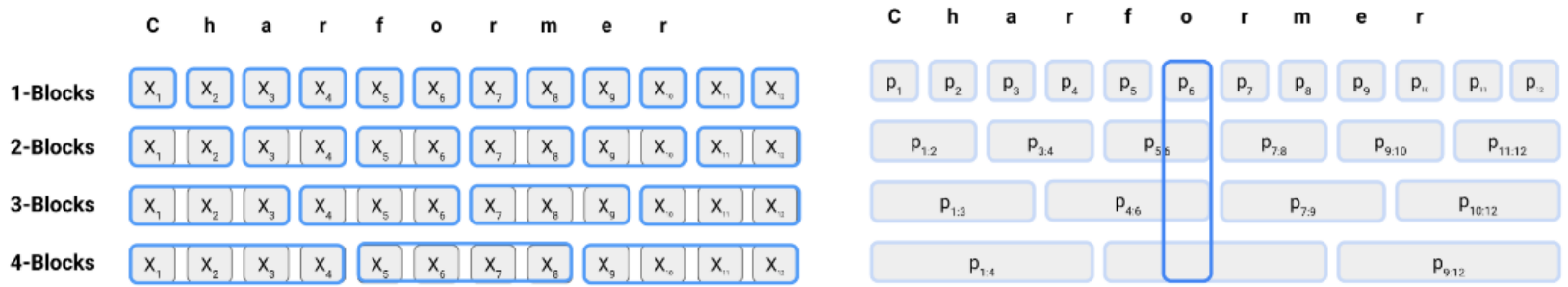
ByT5 – Better at multilingual and noisy input



		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	−24.0	−18.3
	ByT5	**−8.2**	**−19.5**	**−11.4**
Repetitions	mT5	−8.5	−9.5	−12.3
	ByT5	**−4.1**	**−3.0**	**−5.9**
Antspeak	mT5	−32.0	−27.7	−34.4
	ByT5	**−8.7**	**−4.8**	**−24.4**
Uppercase	mT5	−7.0	−8.0	−8.1
	ByT5	**−1.5**	**−0.5**	**−1.7**
Random Case	mT5	−25.7	−14.3	−19.2
	ByT5	**−1.5**	**−0.2**	**−5.9**

Table 6: Degradation of mT5 and ByT5 under various types of noise. "Clean" shows original task performance. Subsequent rows show the delta from "clean" when adding different types of noise. Learnable noise is added in training and eval, while unseen noise only affects eval.

Charformer: A more efficient T5



(a) Formation of subword blocks to be scored by F_R. Offsets and/or pre-GBST convolutions not shown.

(b) Block scores that have been expanded back to length L. Softmax is taken over block scores at each position i to form block weights for constructing latent subword representations.

Figure 2: Illustration of subword block formation and scoring.

Charformer Speed up

Table 6: Pre-training compute metrics of models at different input lengths, downsampling rates, and model sizes on the English C4 dataset. 16 TPUv3 chips were used for this experiment. These numbers reflect a batch size of 64. Memory refers to per-device peak memory usage on TPUv3 chips.

Model	L	d_s	$\|\theta\|$	Speed (steps/s)	FLOPS	Peak Mem.
T5_{Base} (Subword)	512	-	220M	9.3	1.1×10^{13}	-
Byte-level T5_{Base}	1024	1	200M	8.2	2.9×10^{13}	3.09GB
Byte-level T5+LASC_{Base}	1024	4	205M	15	9.9×10^{12}	1.62GB
CHARFORMER_{Base}	1024	2	206M	11	1.6×10^{13}	1.95GB
CHARFORMER_{Base}	1024	3	203M	15	1.1×10^{13}	1.63GB
$\text{CHARFORMER}_{SBase}$	1024	2	134M	14	1.3×10^{13}	1.73GB
$\text{CHARFORMER}_{SBase}$	1024	3	134M	20	8.7×10^{12}	1.34GB

Charformer is still used internally in certain Google products (as of 2023).

Token-free models so far

- Are capable of “encoding” character/bytes efficiently. Have been shown to work well for text understanding tasks
- They are not capable of/efficient at generating text
- They still implicitly tokenize in “fixed length tokens” or patches

Decoder Only Models: MEGABYTE

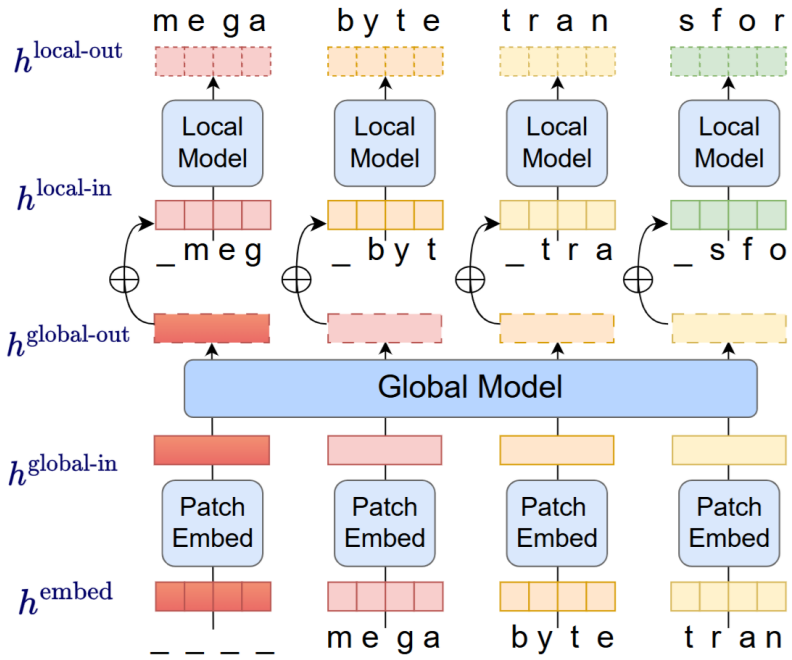


Figure 1. Overview of MEGABYTE with patch size $P = 4$.

Dynamic Patches: *Learning* to Tokenize with the model

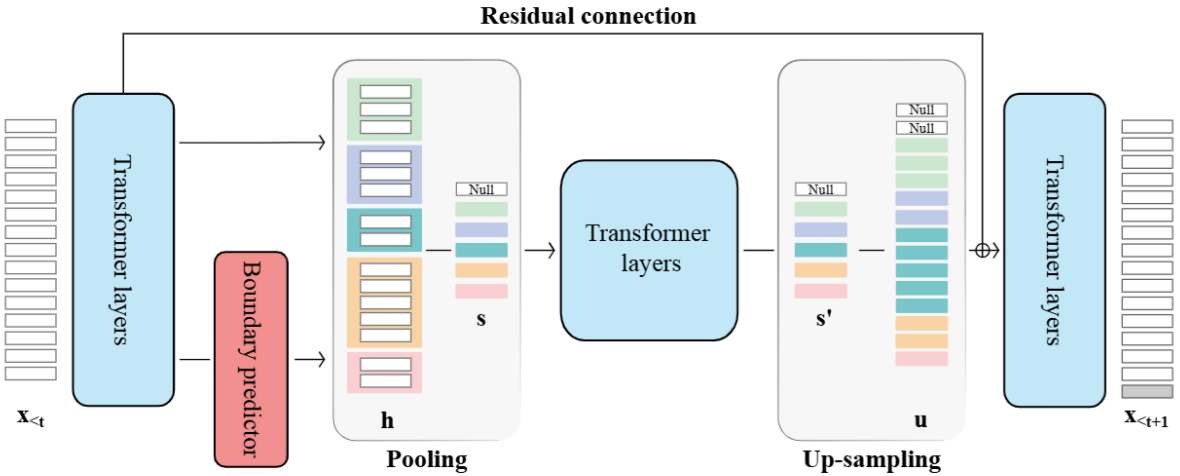


Figure 1: The architecture of a dynamic-pooling Transformer, which jointly performs language modelling and token segmentation. The boundary predictor predicts segment boundaries and pools together groups of variable length by averaging. The shortened sequence is processed efficiently by a series of intermediate layers, then up-sampled back to the original length via duplication. The model generates the next token $\boldsymbol{x}_t$ in the same resolution as the input.

Allows controlling the sequence length via a hyperparameter

	English						Finnish		Hebrew		Vietnamese	
	text8		wiki40b		cc-100		wiki40b		wiki40b		wiki40b	
	BPC	SF	BPC	SF	BPC	SF	BPC	SF	BPC	SF	BPC	SF
Vanilla	1.143	(1.0x)	1.091	(1.0x)	1.225	(1.0x)	0.945	(1.0x)	1.274	(1.0x)	1.065	(1.0x)
Fixed (SF=2)	1.149	(2.0x)	1.084	(2.0x)	1.224	(2.0x)	0.946	(2.0x)	1.279	(2.0x)	1.060	(2.0x)
Fixed (SF=3)	1.155	(3.0x)	1.093	(3.0x)	1.229	(3.0x)	0.951	(3.0x)	1.290	(3.0x)	1.068	(3.0x)
Fixed (SF=4)	1.166	(4.0x)	1.102	(4.0x)	1.240	(4.0x)	0.961	(4.0x)	1.304	(4.0x)	1.087	(4.0x)
Gumbel	1.136*	(4.6x)	1.080	(4.7x)	**1.212***	(4.6x)	0.941	(2.6x)	1.281	**(4.7x)**	1.061	(4.3x)

Follow up Works that improve dynamic patches

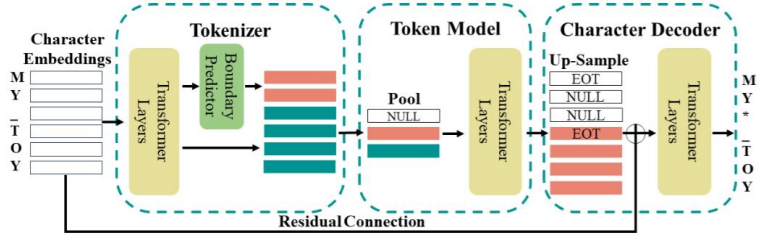


Figure 2: The architecture for Toucan, the token-aware Hourglass Transformer. End-of-token (EOT) vectors and labels (*) are inserted into the character sequences so that the decoder learns token boundaries during training. As per the original model, learned NULL vectors are used to predict the characters in the first token.

Enable faster generation (similar to MEGABYTE)

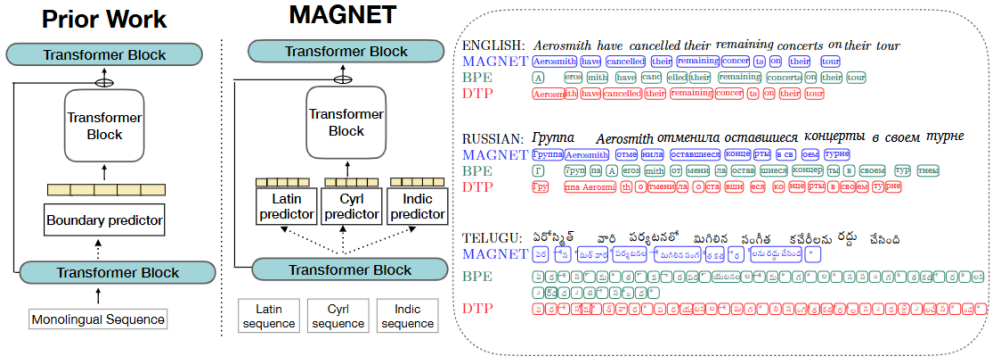


Figure 1: MAGNET routes byte-level sequences via language-script specific boundary predictors. These predictors infer boundaries leading to equitable segmentation across languages. Prior work infers boundaries with a single predictor across languages and leads to over-segmentation.

Enable fairer tokenization rates across languages

Tokenization-Free LM: Promises and Challenges

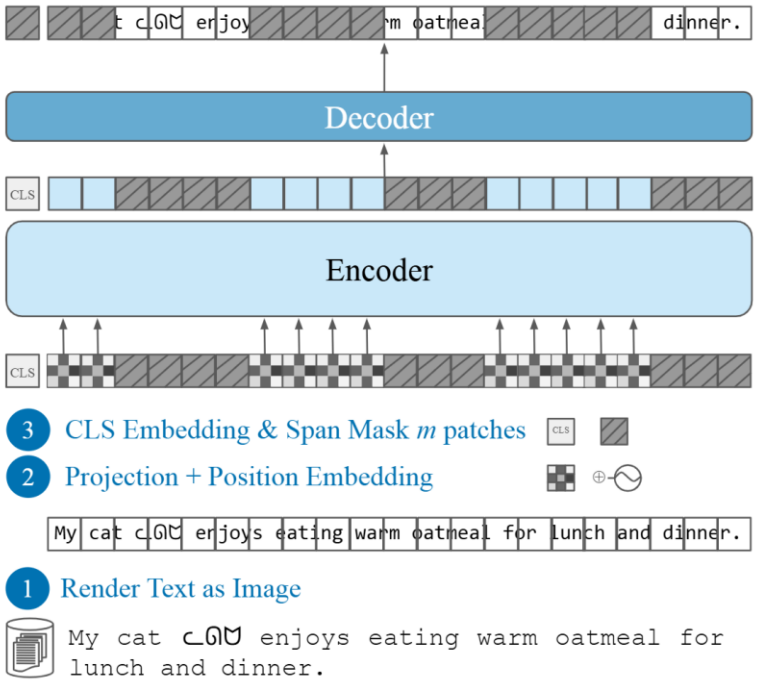
Subword Tokenization

- Produces short sequences of tokens, thus speeds up processing.
- Enables learning local structure: can preserve morphological boundaries.
- Performs better than naive byte/character LM.

Characters or Bytes

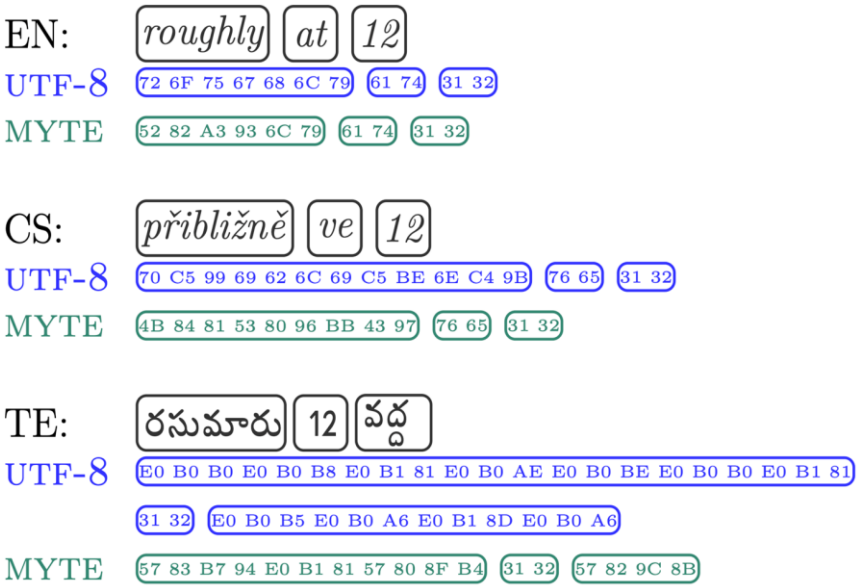
- Data-agnostic, does not require training or deciding on tokenization rules.
- Covers all writing scripts and languages.
- Unaffected by text variation (typos, inflection, dialectical spelling).
- Can encode other modalities.

Bonus: Text as images



Bonus: Redefine Unicode

- MYTE: Morphological Byte. Instead of storing characters, store morphemes as multi-byte sequences



Bonus: Use faster architectures -- MambaByte

- Mamba: An efficient architecture show to perform similarly to Transformers but MUCH faster

- Idea: simply model bytes or characters using a Mamba architecture

What's the outlook?

- New tokenizer-free methods have shown promise especially for multilingual models and noisy inputs.

- What about other issues?
  - Do character based models perform character-level tasks (reverse a string, count the number of characters in a string etc).
  - Have math abilities improved?
  - Have coding abilities improved?
  - ....

Tokenization in other modalities

Why do we need to tokenize image, speech, videos?

To train multimodal LMs. If we can represent each modality as discrete token, we can train a multimodal LM as next token prediction by mixing modalities

Tokenization in other modalities

Speech

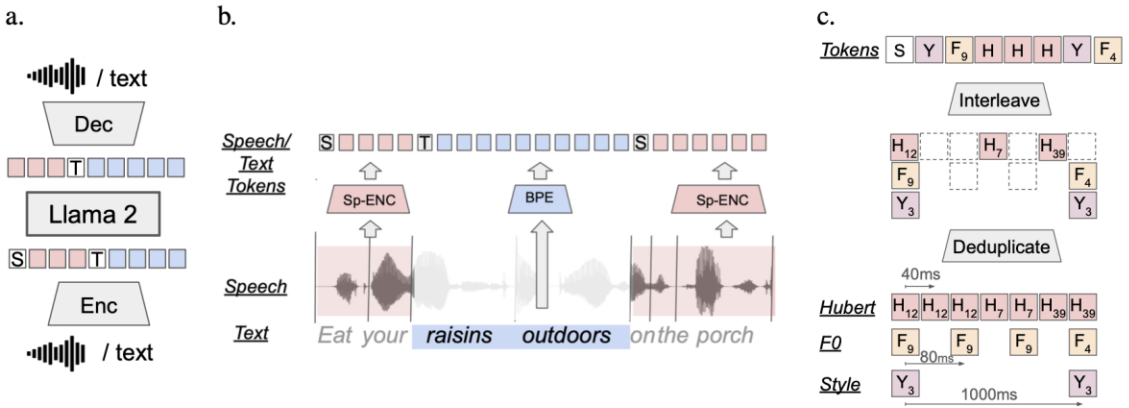
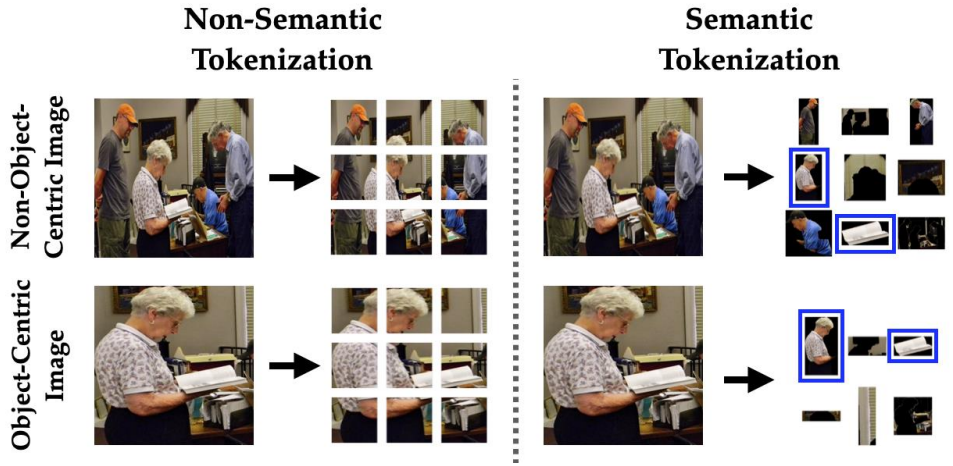


Figure 1: **a. The SPIRIT LM architecture.** A language model trained with next token prediction; tokens are derived from speech or text with an encoder, and rendered back in their original modality with a decoder. SPIRIT LM models are trained on a mix of text-only sequences, speech-only sequences, and *interleaved* speech-text sequences. **b. Speech-text interleaving scheme.** Speech is encoded into tokens (pink) using clusterized speech units (Hubert, Pitch, or Style tokens), and text (blue) using BPE. We use special tokens [TEXT] to prefix text and [SPEECH] for speech tokens. During training, a change of modality is randomly triggered at word boundaries in aligned speech-text corpora. Speech tokens are deduplicated and interleaved with text tokens at the modality change boundary. **c. Expressive Speech tokens.** For SPIRIT LM EXPRESSIVE, pitch tokens and style tokens are interleaved after deduplication.

Tokenization in other modalities

Images



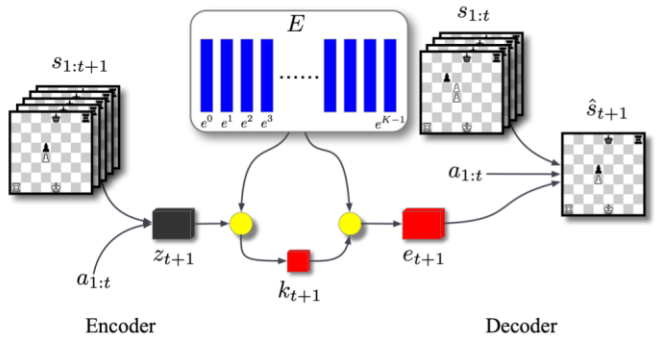


Figure 2. **Complete encoder/decoder architecture of the state VQVAE.** Encoder compresses $s_{1:t+1}$ and $a_{1:t}$ to a continuous latent z_{t+1}. The quantization layer returns the nearest code e_{t+1}, as well as the corresponding index k_{t+1}, in its codebook E. Decoder uses $s_{1:t}$, $a_{1:t}$ and the code $e_{t+1} = E[k_{t+1}]$ to reconstruct s_{t+1}.