Tokenization for Large Language Models

EMNLP 2026 Tutorial

October 28, 2026



Tokenization playground

Presenters



Valentin Hofmann
LMU Munich



Julie Kallini
Stanford University
Remote



Sachin Kumar
Ohio State University



Jindřich Libovický
Charles University



Tomasz Limisiewicz
University of Washington · Meta



Elizabeth Salesky
Google Deepmind

Schedule

9:00 – 9:30	**What is tokenization and why we need it** Jindřich Libovický
9:30 – 10:00	**Subword tokenization** Valentin Hofmann
10:00 – 10:30	**Issues with subword tokenizers** Sachin Kumar
10:30 – 11:00	**Coffee break**
11:00 – 11:15	**Vocabulary expansion** Elizabeth Salesky
11:15 – 11:30	**Impact on Language Models** Tomasz Limisiewicz
11:30 – 12:15	**Bytes & Latent tokenization** Julie Kallini & Tomasz Limisiewicz
12:15 – 12:30	**Visual representations of text** Elizabeth Salesky

Tokenization: A Brief History

Between words and characters: A Brief History of Open-Vocabulary NLP

Sabrina J. Mielke, Zaid Alyafeai et al.

arXiv:2112.10508



Between words and characters: A Brief History of the Open-Vocabulary Approach to Natural Language Processing

Sabrina J. Mielke Zaid Alyafeai Colin Raffel Colin Leong
Victor Sanh Dani Yogatama Graham Neubig ...
(BigScience Tokenization Working Group)

Abstract

The open-vocabulary problem—how to process text containing arbitrary or unseen words—has been a central challenge in natural language processing since its inception. While modern NLP models largely rely on subword tokenization algorithms like Byte-Pair Encoding (BPE), WordPiece, or Unigram Language Modeling, these approaches are the latest in a long history of methods designed to bridge the gap between words and characters.

This survey provides a comprehensive historical account of open-vocabulary modeling in NLP. We trace the lineage of methods from early morphological analysis, character-level n-grams, and subword representations to byte-level modeling and visual representations. We analyze the trade-offs between vocabulary size, sequence length, compute efficiency, and linguistic plausibility, discussing persistent challenges in multilingual settings, noise robustness, and out-of-domain generalization.

Mielke et al. (2021)

20
authors

28
pages

What it covers

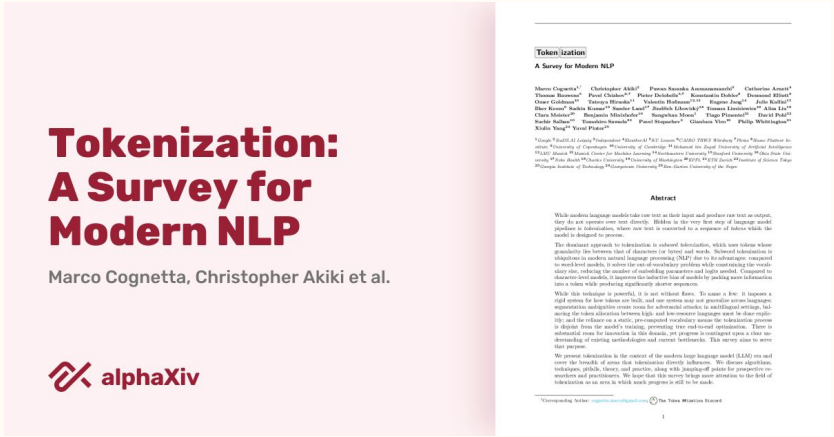
Historical overview of open-vocabulary NLP, tracing evolution from word- and character-level models to subword segmentation and hybrid methods.

From the BigScience tokenization working group.

Read the survey on arXiv

arxiv.org/abs/2112.10508

Tokenization: A Survey for Modern NLP



Cognetta et al. (2026)

32
authors

154
pages

What it covers

Algorithms, evaluation, multilinguality, encodings, theory, and alternatives such as latent or visual tokenization.

Five of today's presenters are co-authors.

Read the survey on alphaXiv

alphaxiv.org/abs/2609.tokenization-survey-modern-nlp

Part I

Introduction, subword tokenization and its woes

Introduction

Why LLMs need tokenization

Discrete units

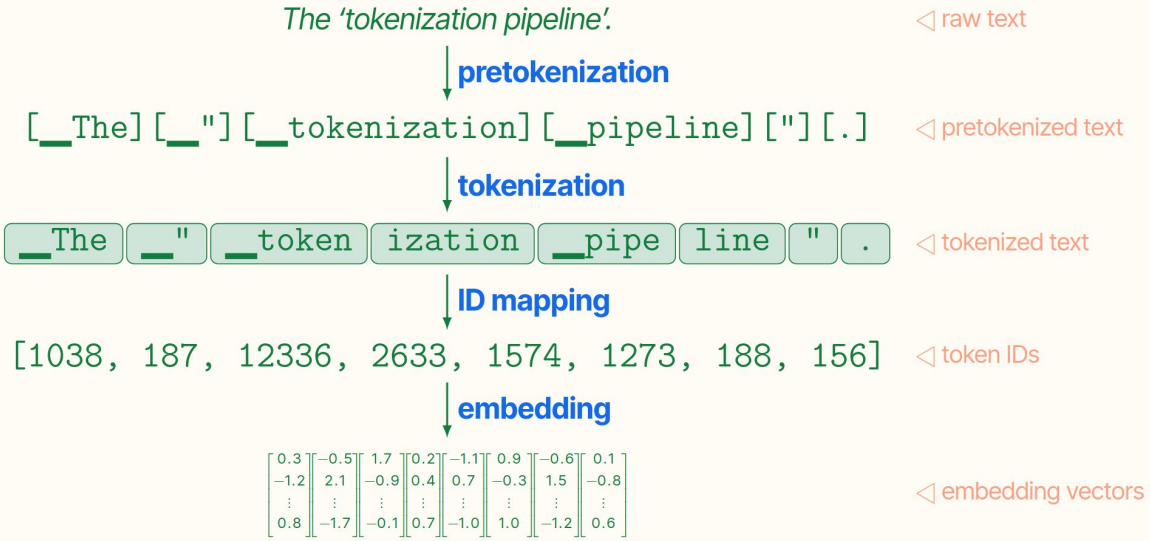
LLMs operate on discrete units, not raw text.

Byte sequences are too long

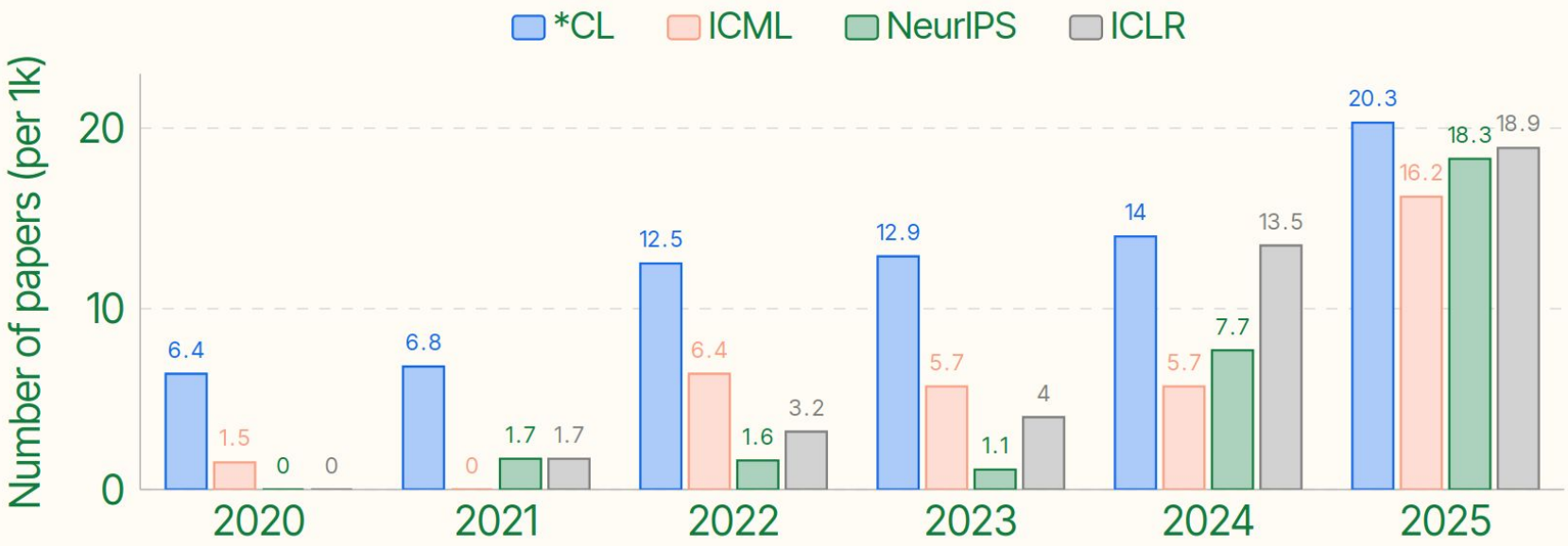
Unless there is a specialized architecture.

Explicit compression layer

Tokenization sits between text and the model.



Research interest



Three challenges of tokenizing languages

1

Scripts and conventions

Different languages use different scripts, and writing conventions are countless.

Next: UTF-8 and scripts

2

Statistics and morphology

Statistics of languages and their morphology complicate things.

Next: Zipf, morphology

3

Uneven data

Different languages have very different amounts of data.

Next: text resources

Terms used in this tutorial

Code point

Unicode assigns each character a number, its code point.

Byte

UTF-8 encodes each code point as 1 to 4 bytes.

Word

A linguistic unit, often found by splitting on whitespace.

Token

One unit of the model's input: a word, subword, character or byte.

Vocabulary

The set of units the tokenizer can produce.

UTF-8

Variable-length encoding of Unicode code points. Older encodings exist but are not used.

Bytes	Code point range	Bit pattern and scripts
1 byte	**U+0000 – U+007F** 128 code points	`0xxxxxxx` ASCII
2 bytes	**U+0080 – U+07FF** 2,048 code points	`110xxxxx 10xxxxxx` Latin ext., Arabic, Hebrew...
3 bytes	**U+0800 – U+FFFF** 63,488 code points	`1110xxxx 10xxxxxx 10xxxxxx` Most CJK, Devanagari, ...
4 bytes	**U+10000 – U+10FFFF** 1,048,576 code points	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx` Emoji, rare scripts

UTF-8 across scripts

Backward compatible
ASCII text keeps its single-byte encoding.

Long byte sequences
Some languages need many more bytes for the same text.

Tokens cross boundaries
A BPE merge can join bytes from different code points.

	Tamil	Sinhala	Devanagari
Grapheme	ன்	සී	वा
Unicode characters	ன + ◌்	ස + ◌ී	व + ◌ा
Unicode code points	U+0BA9, U+0BCD	U+0DC3, U+0DCA	U+0935, U+093E
UTF-8 bytes	e0 ae a9 e0 af 8d	e0 b7 83 e0 b7 8a	e0 a4 b5 e0 a4 be

Same text, different cost

2.6× **Hindi bytes**
31,565 vs 12,333 in English

4.9× **Hindi tokens, Qwen 3**
10,876 vs 2,228 in English

7.4× **Amharic tokens, Llama 3.3**
16,348 vs 2,208 in English

Language	Script	UTF-8		LLM Tokenizers		
		Characters	Bytes	Llama 3.3	Gemma 3	Qwen 3
English	Latin (alphabetic)	12,321	12,333	2,208	2,405	2,228
All human beings are born free and equal in dignity and rights.						
German	Latin (alphabetic)	13,623	13,799	3,462	2,996	3,456
Alle Menschen sind frei und gleich an Würde und Rechten geboren.						
Czech	Latin (alphabetic)	11,518	12,829	3,366	3,629	4,772
Všichni lidé se rodí svobodní a sobě rovní ve své důstojnosti a právech.						
Russian	Cyrillic (alphabetic)	13,493	23,416	3,464	3,133	3,738
Все люди рождаются свободными и равными в своем достоинстве и правах.						
Chinese	Han (logographic)	4,676	10,256	2,693	2,394	2,130
人人生而自由，在尊严和权利上一律平等。						
Korean	Hangul (syllabic)	6,399	13,088	2,932	3,017	3,078
모든 인간은 태어날 때부터 자유로우며 그 존엄과 권리에 있어 평등하다.						
Hindi	Devanagari (syllabic)	13,165	31,565	6,131	3,202	10,876
सभी मनुष्य स्वतंत्र और समान सम्मान और अधिकारों के साथ जन्मे हैं।						
Hebrew	Hebrew (abjad)	8,924	14,709	7,283	3,797	3,066
ובזכויותיהם□ בכבודם ושווים חופשיים נולדו האדם בני כל						
Amharic	Ethiopic (syllabic)	6,393	17,223	16,348	4,934	6,439
ሁሉም ሰዎች ነፃ እና በክብርና መብቶቻቸው ላይ እኩል የተወለዱ ናቸው።						

Tokenization was always there in NLP

Until ~2013

Classical NLP

Texts were split into units inspired by linguistic words.

2013

Word-level models

word2vec: hundreds of thousands of words, with embeddings pre-trained rather than trained with the net.

2016+

Subwords

Subword tokenization takes over. The word-splitting step survives as pre-tokenization.

~~Tokenization~~ → **Pre-tokenization**

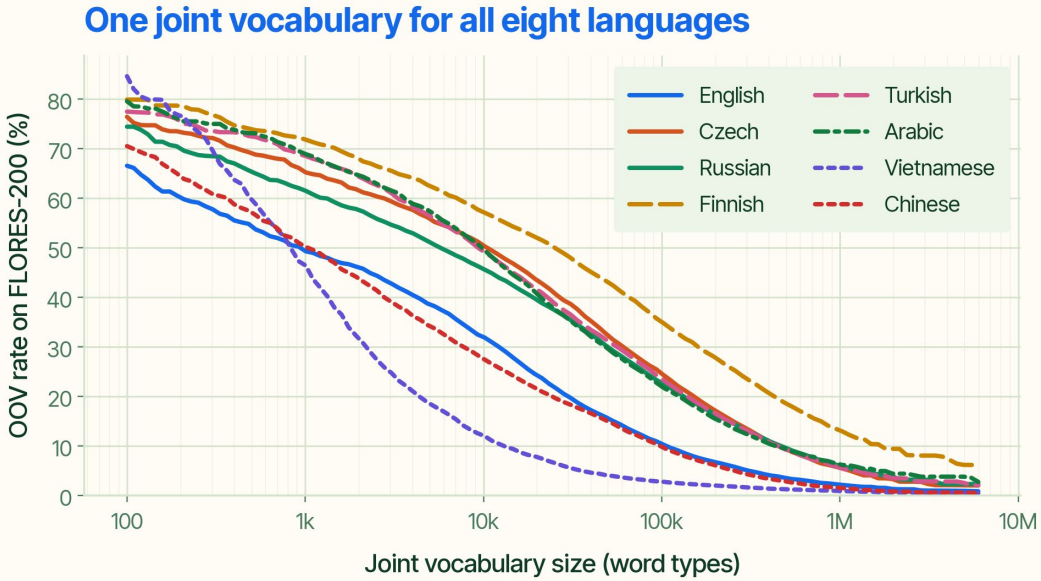
Same step, new name. Typical choices:

- Whitespace
- Punctuation
- Digits
- Language-specific rules

🔗 Pretokenization playground

OOV problem: With words unavoidable

- Now matter of the training data size, you will not see all words
- Typos create unseen word forms
- Language specific: some languages have more word forms

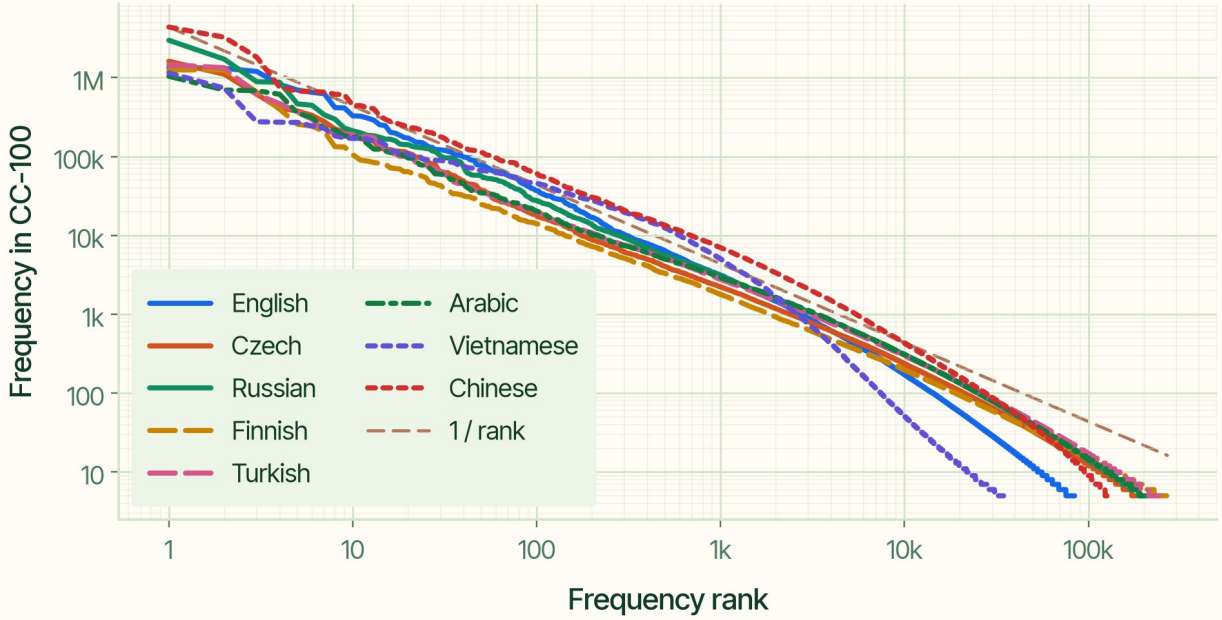


Setup
Vocabulary estimated on 1M sentences of English and Czech (CC100), tested on Flores+.

Colab: vocabulary size vs OOV rate

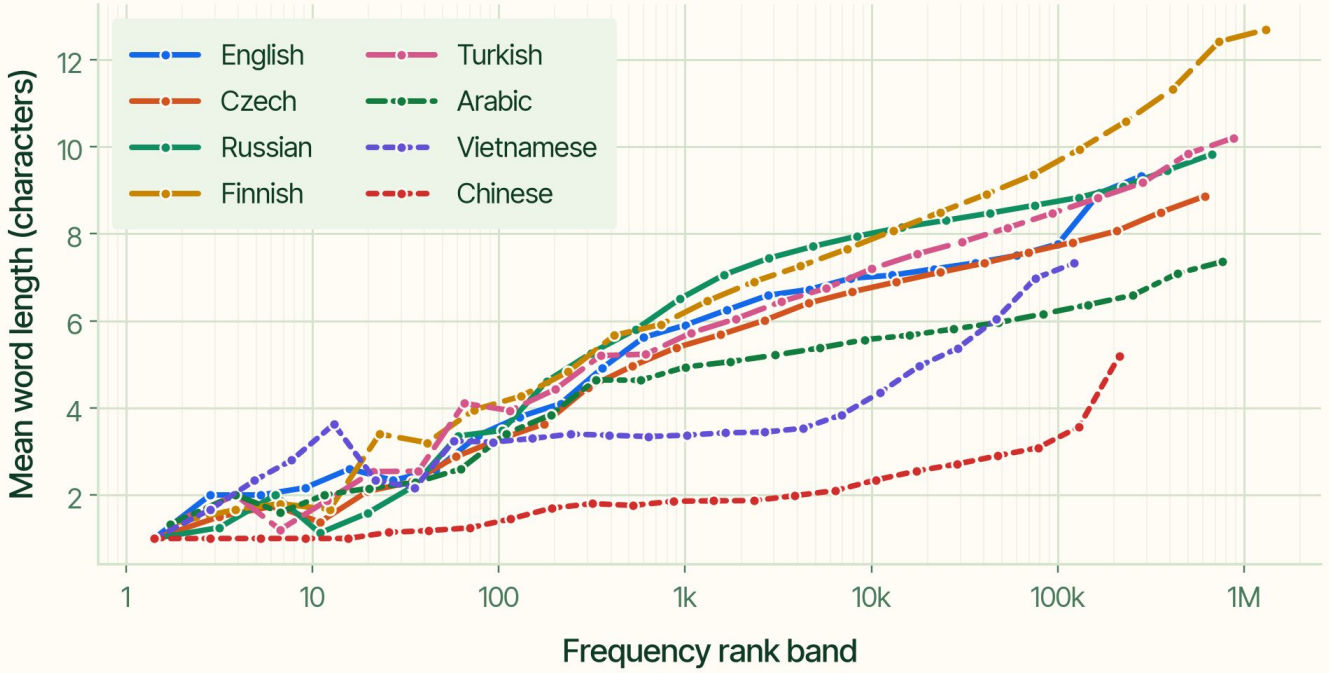
Zipf's law

frequency ∝ 1 / rank



A few words dominate, most are rare • Different languages have different curves

Still Zipf's law: Rarer words are longer words

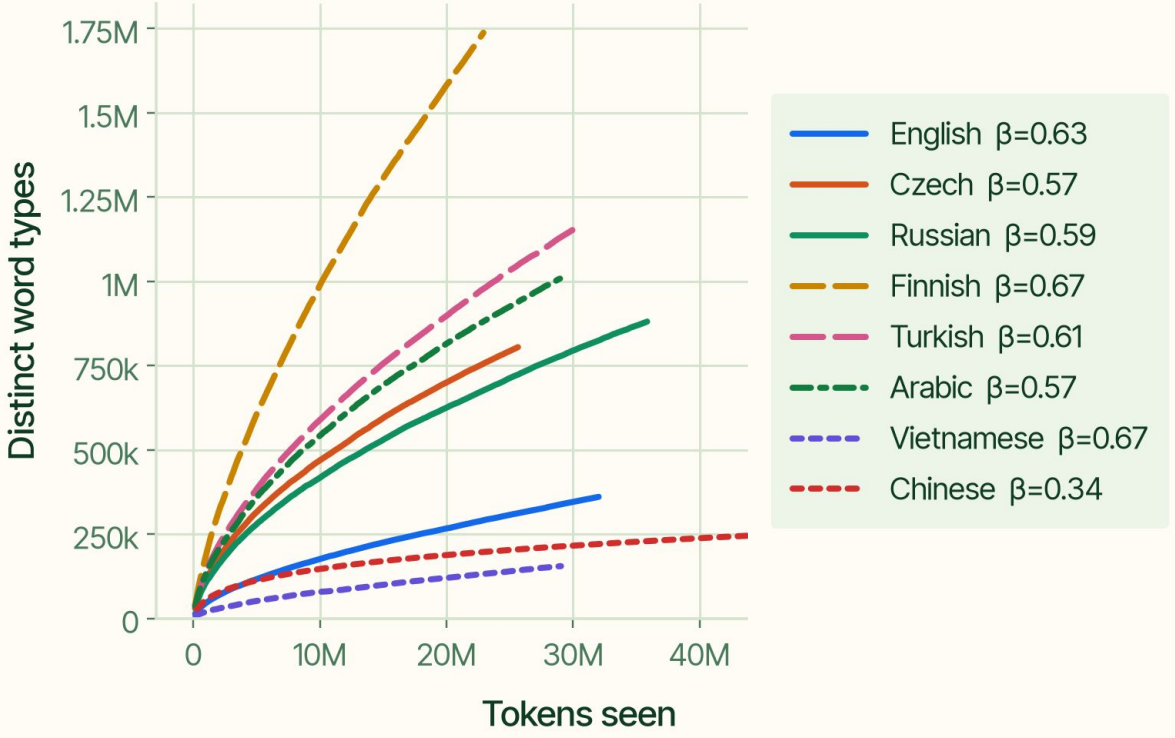


Heap's law

$$\text{vocab size} \propto \text{word seen}^{\beta}$$

Typically $\beta \in [0.4, 0.6]$

Word form vocabulary keeps growing with the training corpus size.



Morphology

Too simplified

Isolating

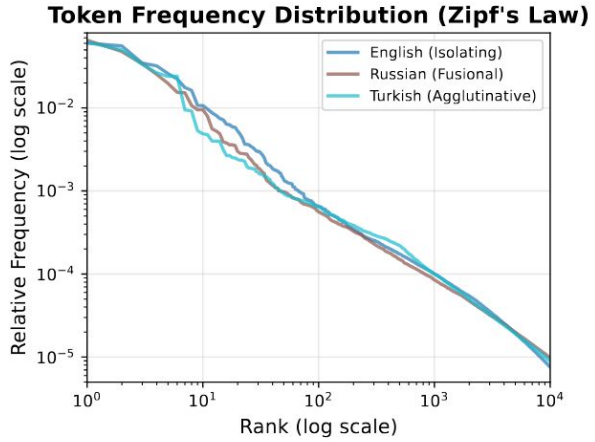
Chinese: mostly one morpheme per word

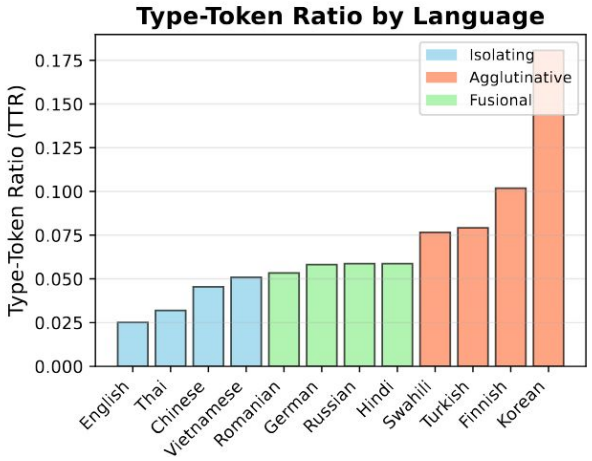
Fusional

Russian: [illegible] several meanings

Agglutinative

Turkish: many affixes, one [illegible]





Why it matters

Morphology shows up in how quickly frequency decreases, and in how many embeddings end up undertrained.

Text resources available

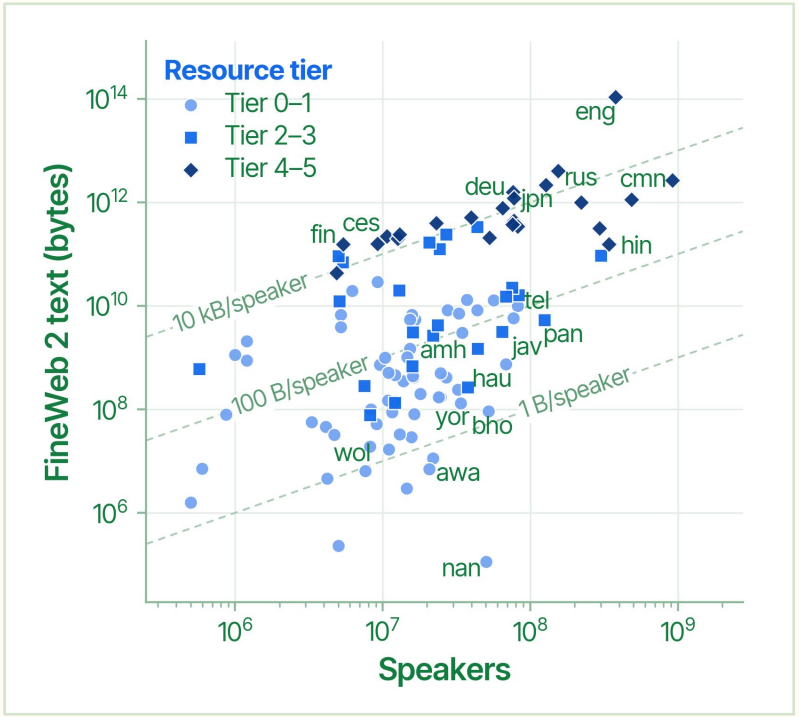
Text data is spread very unevenly across languages

Orders of magnitude

Available text differs by many orders of magnitude between languages.

Vocabulary coverage

With less data, fewer words occur often enough to be well covered by the vocabulary.



Words do not work, so...

We need something that has a more friendly distribution that words and no OOV.

Ideally also, treat languages equally regardless of script, morphology and amount of available resources.

Subword Tokenization

Subword Tokenization

1. General Overview
2. BPE and Unigram
3. Variants

Subword Tokenization

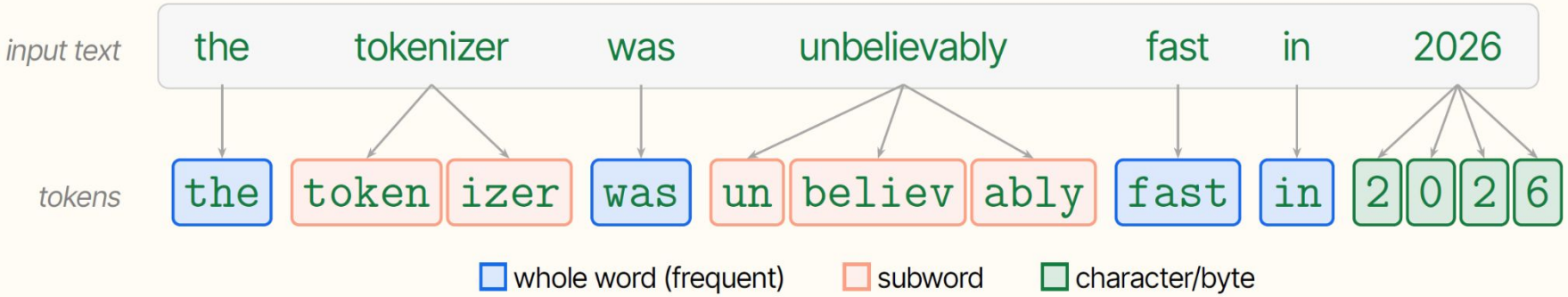
1. General Overview

2. BPE and Unigram

3. Variants

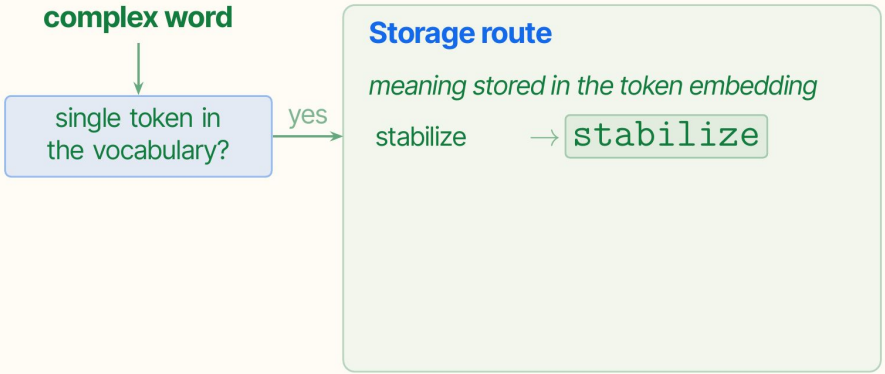
Subword-Level Tokenization

- Idea: split the input text into a sequence of **words**, **subwords**, and **characters/bytes**
- Relatively small token vocabularies (30K to 200K tokens)
- Frequent words represented as wholes, less frequent words represented via subwords and characters/bytes



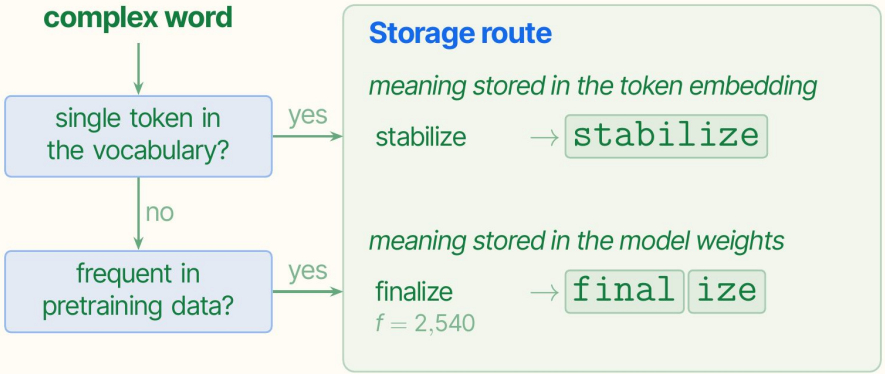
Comparison With Cognitive Models

- Subword-level tokenizers ≈ **dual-route models** (Hofmann et al., 2021)



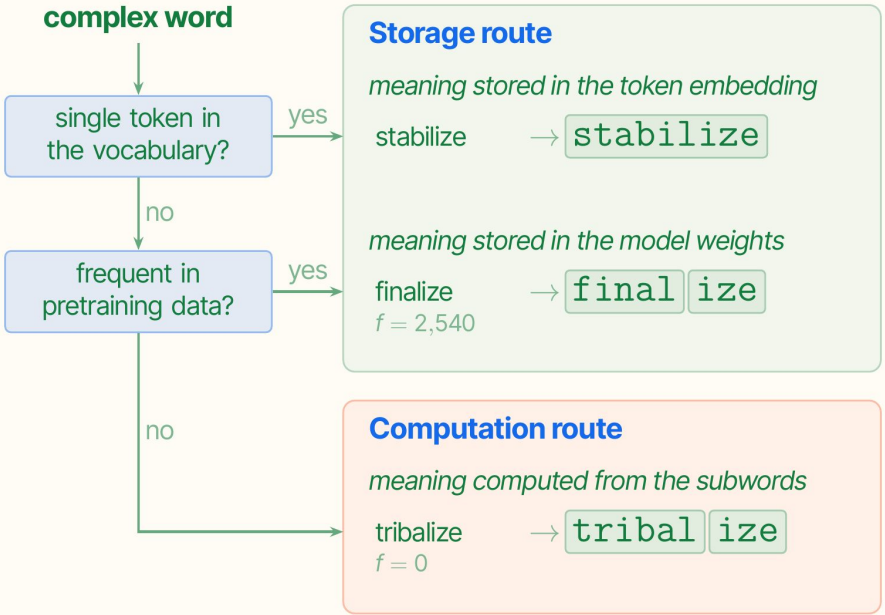
Comparison With Cognitive Models

- Subword-level tokenizers ≈ **dual-route models** (Hofmann et al., 2021)



Comparison With Cognitive Models

- Subword-level tokenizers ≈ **dual-route models** (Hofmann et al., 2021)



Popular Subword-Level Tokenizers

Byte-pair encoding (BPE)
Sennrich et al. (2016)

Unigram
Kudo (2018)

WordPiece
Schuster & Nakajima (2012)

Key differences

1. **Training**: How to learn the token vocabulary and tokenization model?
2. **Inference**: How to map the input text to a sequence of tokens?

Steps Before Actual Subword Tokenization

Normalization

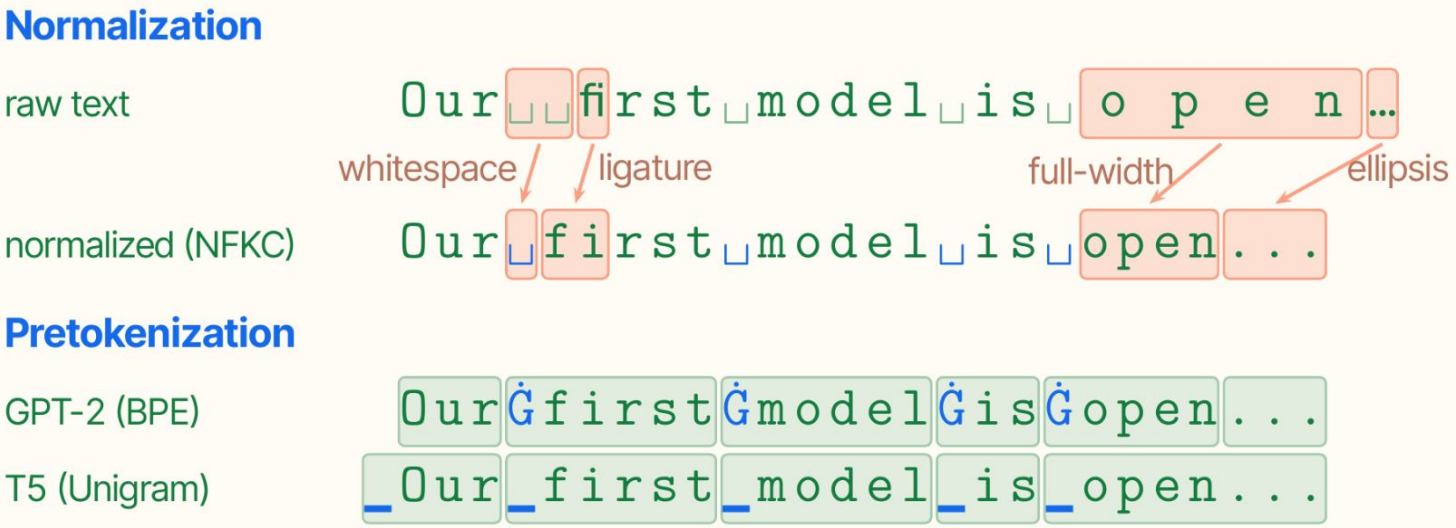
raw text



Steps Before Actual Subword Tokenization



Steps Before Actual Subword Tokenization



Subword Tokenization

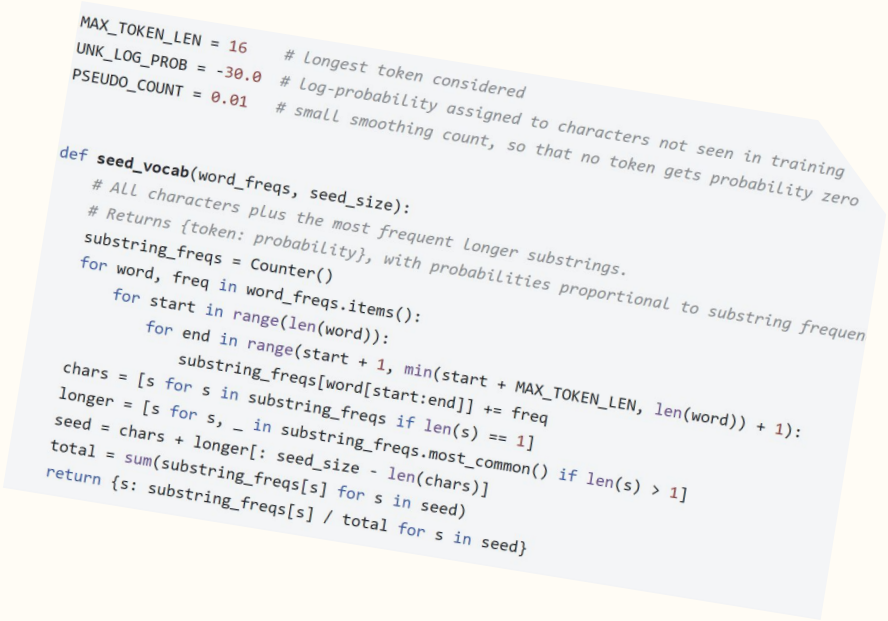
1. General Overview

2. BPE and Unigram

3. Variants

Interactive Demo

https://tokenization-tutorial.github.io/bpe-unigram/



```
MAX_TOKEN_LEN = 16    # longest token considered
UNK_LOG_PROB = -30.0  # log-probability assigned to characters not seen in training
PSEUDO_COUNT = 0.01   # small smoothing count, so that no token gets probability zero


def seed_vocab(word_freqs, seed_size):
    # All characters plus the most frequent longer substrings.
    # Returns {token: probability}, with probabilities proportional to substring frequen
    substring_freqs = Counter()
    for word, freq in word_freqs.items():
        for start in range(len(word)):
            for end in range(start + 1, min(start + MAX_TOKEN_LEN, len(word)) + 1):
                substring_freqs[word[start:end]] += freq
    chars = [s for s in substring_freqs if len(s) == 1]
    longer = [s for s, _ in substring_freqs.most_common() if len(s) > 1]
    seed = chars + longer[: seed_size - len(chars)]
    total = sum(substring_freqs[s] for s in seed)
    return {s: substring_freqs[s] / total for s in seed}
```

BPE Overview

- Originally a **text compression algorithm** proposed by Gage (1994)
- Adapted to tokenization by Sennrich et al. (2026) [**ACL 2026 Test of Time Award**]
- Most widely used tokenizer today: GPT, Llama, Olmo, Qwen, ...

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

BPE Training Pseudocode

```
Input: set of strings D, target vocab size k
procedure BPE(D, k)
    V ← all unique characters in D
    while |V| < k do
        t_L, t_R ← Most frequent bigram in D
        t_new ← t_L + t_R
        V ← V + [t_new]
        Replace each occurrence of t_L, t_R in D with t_new
    end while
    return V
end procedure
```

BPE Training Pseudocode

Create **initial vocabulary consisting of all characters/bytes** in corpus



```
1:  Input: set of strings D, target vocab size k
2:  procedure BPE(D, k)
3:      V ← all unique characters in D
4:      while |V| < k do
5:          t_L, t_R ← Most frequent bigram in D
6:          t_new ← t_L + t_R
7:          V ← V + [t_new]
8:          Replace each occurrence of t_L, t_R in D with t_new
9:      end while
10:     return V
11: end procedure
```

BPE Training Pseudocode

Create **initial vocabulary consisting of all characters/bytes** in corpus

Iteratively merge **pair of adjacent tokens with highest frequency**

```
1:  Input: set of strings D, target vocab size k
2:  procedure BPE(D, k)
3:      V ← all unique characters in D
4:      while |V| < k do
5:          t_L, t_R ← Most frequent bigram in D
6:          t_new ← t_L + t_R
7:          V ← V + [t_new]
8:          Replace each occurrence of t_L, t_R in D with t_new
9:      end while
10:     return V
11: end procedure
```

BPE Training Pseudocode

Stop when vocabulary reaches **predefined size**

Create **initial vocabulary consisting of all characters/bytes** in corpus

Iteratively merge **pair of adjacent tokens with highest frequency**

```
1:  Input: set of strings D, target vocab size k
2:  procedure BPE(D, k)
3:      V ← all unique characters in D
4:      while |V| < k do
5:          t_L, t_R ← Most frequent bigram in D
6:          t_new ← t_L + t_R
7:          V ← V + [t_new]
8:          Replace each occurrence of t_L, t_R in D with t_new
9:      end while
10:     return V
11: end procedure
```

BPE Training Demo

Corpus

h u g	×10
p u g	×5
p u n	×12
b u n	×4
h u g s	×5

BPE Training Demo

Corpus

h u g	×10
p u g	×5
p u n	×12
b u n	×4
h u g s	×5

Pair counts

u + g	20
p + u	17
u + n	16
h + u	15
g + s	5
b + u	4

Vocabulary

target $k = 10$, current $|V| = 7$

b g h n p s u

Merge rules (in order)

(none yet)

BPE Training Demo

Corpus

h u g	×10
p u g	×5
p u n	×12
b u n	×4
h u g s	×5

Pair counts

u + g	20
p + u	17
u + n	16
h + u	15
g + s	5
b + u	4

Vocabulary

target $k = 10$, current $|V| = 7$

b g h n p s u

Merge rules (in order)

(none yet)

BPE Training Demo

Corpus

h ug	×10
p ug	×5
p u n	×12
b u n	×4
h ug s	×5

Pair counts

u + n	16
h + ug	15
p + u	12
p + ug	5
ug + s	5
b + u	4

Vocabulary

target $k = 10$, current $|V| = 8$

b g h n p s u ug

Merge rules (in order)

1. u + g → ug

BPE Training Demo

Corpus

h ug	×10
p ug	×5
p u n	×12
b u n	×4
h ug s	×5

Pair counts

u + n	16
h + ug	15
p + u	12
p + ug	5
ug + s	5
b + u	4

Vocabulary

target $k = 10$, current $|V| = 8$

b g h n p s u ug

Merge rules (in order)

1. u + g → ug

BPE Training Demo

Corpus

h ug	×10
p ug	×5
p un	×12
b un	×4
h ug s	×5

Pair counts

h + ug	15
p + un	12
p + ug	5
ug + s	5
b + un	4

Vocabulary

target $k = 10$, current $|V| = 9$

b g h n p s u ug un

Merge rules (in order)

1. u + g → ug
2. u + n → un

BPE Training Demo

Corpus

h ug	×10
p ug	×5
p un	×12
b un	×4
h ug s	×5

Pair counts

h + ug	15
p + un	12
p + ug	5
ug + s	5
b + un	4

Vocabulary

target $k = 10$, current $|V| = 9$

b g h n p s u ug un

Merge rules (in order)

1. u + g → ug
2. u + n → un

BPE Training Demo

Corpus

hug	×10
p ug	×5
p un	×12
b un	×4
hug s	×5

Pair counts

p + un	12
p + ug	5
hug + s	5
b + un	4

Vocabulary

target $k = 10$, current $|V| = 10$

b g h n p s u ug un hug

Merge rules (in order)

1. u + g → ug
2. u + n → un
3. h + ug → hug

BPE Inference

- Split each word into its characters/bytes
- Follow the merge rules found during training

BPE Inference Demo

Learned merge rules

1. `u` + `g` → `ug`
2. `u` + `n` → `un`
3. `h` + `ug` → `hug`

New words

	unhug	bugs	puns
characters	`u` `n` `h` `u` `g`	`b` `u` `g` `s`	`p` `u` `n` `s`

BPE Inference Demo

Learned merge rules

1. u + g → ug
2. u + n → un
3. h + ug → hug

New words

	unhug	bugs	puns
characters	u n h u g	b u g s	p u n s
after rule 1	u n h ug	b ug s	p u n s

BPE Inference Demo

Learned merge rules

1. u + g → ug
2. u + n → un
3. h + ug → hug

New words

	unhug	bugs	puns
characters	u n h u g	b u g s	p u n s
after rule 1	u n h ug	b ug s	p u n s
after rule 2	un h ug	b ug s	p un s

BPE Inference Demo

Learned merge rules

1. u + g → ug
2. u + n → un
3. h + ug → hug

New words

	unhug	bugs	puns
characters	u n h u g	b u g s	p u n s
after rule 1	u n h ug	b ug s	p u n s
after rule 2	un h ug	b ug s	p un s
after rule 3	un hug	b ug s	p un s

Unigram Overview

- Probabilistic approach to tokenization proposed by Kudo (2018)
- Unigram **minimizes the total code length (entropy coding)** vs. total number of tokens in the case of BPE (dictionary coding)
- Used by several language models: XLNet, T5, ALBERT, ...



Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates

Taku Kudo
Google, Inc.
taku@google.com

A Note on SentencePiece

SentencePiece

CI for general build passing | Build Wheels passing | issues 3 open | python 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | 3.14 | pypi package 0.2.2
downloads 27M/month | contributions welcome | License Apache 2.0 | SLSA level 3 | docs google.github.io/sentencepiece

SentencePiece is a fast, lightweight, and unsupervised text tokenizer and detokenizer designed for neural network-based text generation systems (such as Large Language Models) where the vocabulary size is fixed prior to training.

It implements **subword units**—including **Byte-Pair-Encoding (BPE)** [Sennrich et al.] and the **unigram language model** [Kudo.]—with the ability to train directly from raw sentences. By treating input text as a raw sequence of Unicode characters, SentencePiece enables a purely end-to-end, language-independent pipeline that completely eliminates the need for language-specific pre- or post-processing.

Clara Meister's Unigram Manual



https://cimeister.github.io/blog/unigramlm/

Unigram Inference

- For any particular word w, there is typically a **large number of possible segmentations t** = (t_1, t_2, ..., t_m) given a tokenizer vocabulary

w = `unable`

segmentation t

`un` `able`
`una` `ble`
`un` `a` `ble`
`un` `ab` `le`

+ 11 more segmentations

Unigram Inference

- For any particular word w, there is typically a **large number of possible segmentations t** = (t_1, t_2, ..., t_m) given a tokenizer vocabulary
- Unigram assigns each token t_i in the vocabulary a probability $P(t_i)$
- We pick the segmentation with the highest joint probability (**Viterbi algorithm**)

w = `unable`

segmentation t	$\prod_i P(t_i)$	$P(\mathbf{t})$
`un` `able`	$0.05 \cdot 0.05$	$= 2.5 \times 10^{-3}$
`una` `ble`	$0.0025 \cdot 0.018$	$= 4.5 \times 10^{-5}$
`un` `a` `ble`	$0.05 \cdot 0.018 \cdot 0.018$	$= 1.6 \times 10^{-5}$
`un` `ab` `le`	$0.05 \cdot 0.0067 \cdot 0.018$	$= 6.0 \times 10^{-6}$

+ 11 more segmentations

Unigram Training Basics

- Given a training corpus D, we need to find (i) the vocabulary of tokens V and (ii) the token probabilities $P(t_i)$

Unigram Training Basics

- Given a training corpus D, we need to find (i) the vocabulary of tokens V and (ii) the token probabilities $P(t_i)$
- Unigram uses a **top-down** approach: we start with a large initial vocabulary and **iteratively prune tokens** until we reach the desired vocabulary size, while at the same time learning token probabilities

Unigram Training Basics

- Given a training corpus D, we need to find (i) the vocabulary of tokens V and (ii) the token probabilities $P(t_i)$
- Unigram uses a **top-down** approach: we start with a large initial vocabulary and **iteratively prune tokens** until we reach the desired vocabulary size, while at the same time learning token probabilities
- Guiding objective: maximize **observed-data log likelihood** of the training corpus

$$P(w) = \sum_{\mathbf{t} \in S(w)} P(\mathbf{t}) = \sum_{\mathbf{t} \in S(w)} \prod_{i=1}^{m} P(t_i)$$

marginal probability of a word

$$\mathcal{L} = \sum_{w \in \mathcal{D}} \text{freq}(w) \cdot \log P(w)$$

observed-data log likelihood

Initialization

- We start with a **large initial vocabulary** (around 1M tokens)
- Typically: bytes/characters, all substrings up to some length, frequent words

Initialization

- We start with a **large initial vocabulary** (around 1M tokens)
- Typically: bytes/characters, all substrings up to some length, frequent words
- Token probabilities **initialized proportional to frequency** in training corpus

Initialization

- We start with a **large initial vocabulary** (around 1M tokens)
- Typically: bytes/characters, all substrings up to some length, frequent words
- Token probabilities **initialized proportional to frequency** in training corpus
- Choice matters: if token not in initial vocabulary, it will never be part of the tokenizer!

Learning Token Probabilities

- Maximizing the observed-data log likelihood is difficult, so we maximize the **complete-data log likelihood** instead (see Unigram Manual for details!)
- Method: **expectation-maximization (EM)**

Learning Token Probabilities

- Maximizing the observed-data log likelihood is difficult, so we maximize the **complete-data log likelihood** instead (see Unigram Manual for details!)
- Method: **expectation-maximization (EM)**
- **E-step**: given current probability estimates, compute **expected counts** for each token across all possible segmentations (**forward-backward algorithm**)

$$c(t) = \sum_{w \in \mathcal{D}} \text{freq}(w) \cdot \sum_{\mathbf{t} \in S(w)} P(\mathbf{t} \mid w) \cdot \mathbb{1}[t \in \mathbf{t}]$$

E-step: expected counts

Learning Token Probabilities

- Maximizing the observed-data log likelihood is difficult, so we maximize the **complete-data log likelihood** instead (see Unigram Manual for details!)
- Method: **expectation-maximization (EM)**
- **E-step**: given current probability estimates, compute **expected counts** for each token across all possible segmentations (**forward-backward algorithm**)
- **M-step**: given expected counts from E-step, update probability estimates for all tokens using **maximum likelihood estimation**

$$c(t) = \sum_{w \in \mathcal{D}} \text{freq}(w) \cdot \sum_{\mathbf{t} \in S(w)} P(\mathbf{t} \mid w) \cdot \mathbb{1}[t \in \mathbf{t}]$$

E-step: expected counts

$$P(t) = \frac{c(t)}{\sum_{t' \in V} c(t')}$$

M-step: token probabilities

Vocabulary Pruning

- After a certain number of EM iterations, we **remove a fraction of vocabulary tokens** (typically 10% to 20%)
- We remove tokens with **lowest impact on the observed-data log likelihood**

$$\Delta\mathcal{L}(t) = \mathcal{L}(V) - \mathcal{L}(V \setminus \{t\})$$

Unigram Training Pseudocode

Input: set of strings D, target vocab size k
procedure UnigramLM(D, k)
    $V \leftarrow$ all substrings occurring more than
        once in D (not crossing words)
    while $|V| > k$ **do**
        Fit unigram LM θ to D
        for $t \in V$ **do**
            $\Delta\mathcal{L}(t) \leftarrow \log p_\theta(D) - \log p_{\theta'}(D)$
            where θ' is the LM without token t
        end for
        Remove $\min(|V| - k, \lfloor \alpha |V| \rfloor)$ of the
        tokens t with lowest $\Delta\mathcal{L}(t)$ from V,
        where $\alpha \in [0, 1]$ is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure

Unigram Training Pseudocode

Create **initial vocabulary**

```
Input: set of strings D, target vocab size k
procedure UnigramLM(D, k)
    V ← all substrings occurring more than
          once in D (not crossing words)
    while |V| > k do
        Fit unigram LM θ to D
        for t ∈ V do
            Δℒ(t) ← log p_θ(D) − log p_θ′(D)
            where θ′ is the LM without token t
        end for
        Remove min(|V| − k, ⌊α|V|⌋) of the
        tokens t with lowest Δℒ(t) from V,
        where α ∈ [0, 1] is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure
```

Unigram Training Pseudocode

Create **initial vocabulary**

Iteratively **learn token probabilities** using EM

```
Input: set of strings D, target vocab size k
procedure UnigramLM(D, k)
    V ← all substrings occurring more than
        once in D (not crossing words)
    while |V| > k do
        Fit unigram LM θ to D
        for t ∈ V do
            ΔL(t) ← log p_θ(D) − log p_θ'(D)
            where θ' is the LM without token t
        end for
        Remove min(|V| − k, ⌊α|V|⌋) of the
        tokens t with lowest ΔL(t) from V,
        where α ∈ [0, 1] is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure
```

Unigram Training Pseudocode

Create **initial vocabulary**

Iteratively **learn token probabilities** using EM

Iteratively remove tokens with **lowest impact on corpus log likelihood**

```
Input: set of strings D, target vocab size k
procedure UnigramLM(D, k)
    V ← all substrings occurring more than
        once in D (not crossing words)
    while |V| > k do
        Fit unigram LM θ to D
        for t ∈ V do
            Δℒ(t) ← log p_θ(D) − log p_θ′(D)
            where θ′ is the LM without token t
        end for
        Remove min(|V| − k, ⌊α|V|⌋) of the
        tokens t with lowest Δℒ(t) from V,
        where α ∈ [0, 1] is a hyperparameter
    end while
    Fit final unigram LM θ to D
    return V, θ
end procedure
```

Unigram Training Pseudocode

Create **initial vocabulary**

```
1:  Input: set of strings D, target vocab size k
2:  procedure UnigramLM(D, k)
3:      V ← all substrings occurring more than
4:            once in D (not crossing words)
5:      while |V| > k do
6:          Fit unigram LM θ to D
7:          for t ∈ V do
8:              ΔL(t) ← log p_θ(D) − log p_θ'(D)
9:              where θ' is the LM without token t
10:         end for
11:         Remove min(|V| − k, ⌊α|V|⌋) of the
12:         tokens t with lowest ΔL(t) from V,
13:         where α ∈ [0, 1] is a hyperparameter
14:     end while
15:     Fit final unigram LM θ to D
16:     return V, θ
17: end procedure
```

Iteratively **learn token probabilities** using EM

Stop when vocabulary reaches **predefined size**

Iteratively remove tokens with **lowest impact on corpus log likelihood**

Unigram Training Demo

Corpus

best segmentation

hug	×10
pug	×5
pun	×12
bun	×4
hugs	×5

Characters

	$p(t)$
b	
g	
h	
n	
p	
s	
u	

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu		
hug		
bu		
hu		
un		
ug		
gs		
ugs		

Unigram Training Demo

Corpus

best segmentation

hug	×10
pu g	×5
pu n	×12
bu n	×4
hug s	×5

Characters

	$p(t)$
b	0.014
g	0.078
h	0.014
n	0.219
p	0.014
s	0.076
u	0.013

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu	0.232	
hug	0.205	
bu	0.064	
hu	0.015	
un	0.014	
ug	0.014	
gs	0.014	
ugs	0.014	

Unigram Training Demo

Corpus

best segmentation

hug	×10
pu g	×5
pu n	×12
bu n	×4
hug s	×5

Characters

	$p(t)$
b	0.014
g	0.078
h	0.014
n	0.219
p	0.014
s	0.076
u	0.013

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu	0.232	86.05
hug	0.205	67.43
bu	0.064	16.36
hu	0.015	0.15
un	0.014	0.10
ug	0.014	0.07
gs	0.014	0.07
ugs	0.014	0.06

Unigram Training Demo

Corpus

best segmentation

hug	×10
pu g	×5
pu n	×12
bu n	×4
hug s	×5

Characters

	$p(t)$
b	0.014
g	0.078
h	0.014
n	0.219
p	0.014
s	0.076
u	0.013

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu	0.232	86.05
hug	0.205	67.43
bu	0.064	16.36
hu	0.015	0.15
un	0.014	0.10
ug	0.014	0.07
gs	0.014	0.07
ugs	0.014	0.06

Unigram Training Demo

Corpus

best segmentation

hug	×10
pu g	×5
pu n	×12
bu n	×4
hug s	×5

Characters

	$p(t)$
b	0.014
g	0.082
h	0.014
n	0.228
p	0.014
s	0.081
u	0.014

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu	0.242	100.00
hug	0.215	77.70
bu	0.067	16.34
hu	0.015	0.08
un	0.015	0.10
ug		removed
gs		removed
ugs		removed

Unigram Training Demo

Corpus

best segmentation

hug	×10
pu g	×5
pu n	×12
bu n	×4
hug s	×5

Characters

	$p(t)$
b	0.014
g	0.083
h	0.014
n	0.236
p	0.014
s	0.083
u	0.014

Candidate tokens

	$p(t)$	$\Delta\mathcal{L}(t)$
pu	0.250	
hug	0.222	
bu	0.069	
hu		removed
un		removed
ug		removed
gs		removed
ugs		removed

Conceptual Comparison of BPE and Unigram

	BPE	Unigram
Search procedure	bottom-up	top-down
Optimization objective	compression	log-likelihood
Inference	single best	best / sampled

Empirical Comparison of BPE and Unigram

- Unigram leads to segmentations that **more closely match morphological structure** than BPE (e.g., Bostrom and Durrett, 2020)
- Results regarding downstream performance are mixed (e.g., Ali et al., 2024)

Original:	furiously	**Original:**	tricycles	**Original:**	nanotechnology
BPE:	_fur iously	**BPE:**	_t ric y cles	**BPE:**	_n an ote chn ology
Uni. LM:	_fur ious ly	**Uni. LM:**	_tri cycle s	**Uni. LM:**	_nano technology

Original:	Completely preposterous suggestions
BPE:	_Comple t ely _prep ost erous _suggest ions
Unigram LM:	_Complete ly _pre post er ous _suggestion s

Algorithm and Library	DE	FR	IT	ES	EN	AVG	EN
BPE-HF	35.69	37.31	39.37	42.28	47.48	40.43	48.98
BPE-SP	**37.13**	37.45	**40.04**	41.96	**47.68**	**40.85**	49.77
UNI-SP	36.51	**37.66**	39.57	**42.56**	47.10	40.68	**50.01**

Subword Tokenization

1. General Overview
2. BPE and Unigram
3. **Variants**

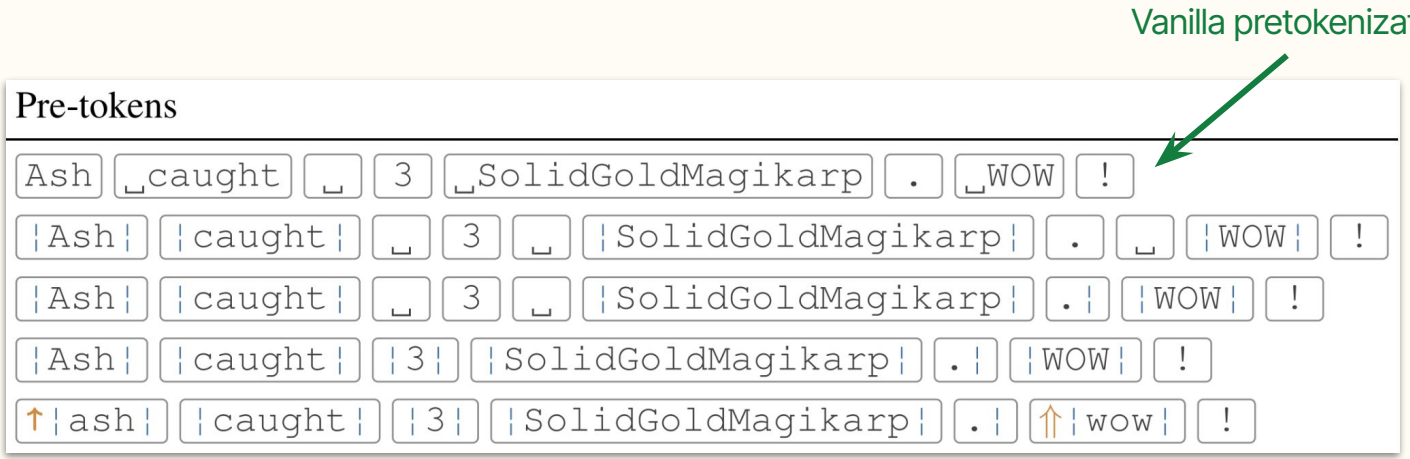
Impact of Normalization

- Normalization (or lack thereof) can have have unintended consequences on model performance (Gorman and Pinter, 2025)
- Example: treatment of **diacritics** (Inoue et al., 2026)

Model	ArabicMMLU ($n = 14,455$)			ArabCulture ($n = 3,482$)			AraTrust ($n = 522$)			Macro Average		
	undiac	wild	full*	undiac	wild	full*	undiac	wild	full*	undiac	wild	full*
AceGPT-v2 Chat (8B)	57.2†	57.0	46.1†	35.3	35.4	34.4	68.8	69.2	58.8†	53.8	53.9	46.4
ALLaM (7B)	70.0	70.0	64.2†	37.1	37.2	35.3†	83.3	83.5	77.0†	63.5	63.6	58.9
Fanar Instruct (9B)	65.3	65.3	55.6†	44.7†	44.9	42.4†	85.4	85.4	78.0†	65.1	65.2	58.6
Jais Chat (13B)	58.2	58.1	46.3†	42.3†	42.7	40.0†	78.7	79.5	56.5†	59.8	60.1	47.6
Jais 2 Chat (8B)	58.6	58.7	50.9†	36.9	36.8	37.0	73.6	73.6	63.4†	56.4	56.4	50.4
Aya-Expanse (8B)	58.9†	59.2	50.7†	35.0	34.9	32.9†	85.6	85.4	76.4†	59.8	59.9	53.3
Llama-3.1 Instruct (8B)	56.2†	56.4	42.5†	35.4	35.3	33.4†	83.3	83.5	62.1†	58.3	58.4	46.0
Qwen3 (8B)	62.5	62.4	56.6†	35.0	35.0	33.3†	85.1	84.7	83.1	60.9	60.7	57.7
GPT-4o	**80.7**	**80.8**	**79.4**†	**88.2**	**88.3**	**87.2**†	**92.9**	**92.5**	**92.9**	**87.3**	**87.2**	**86.5**
Model Average	63.1	63.1	54.7	43.3	43.4	41.8	81.9	81.9	72.0	62.7	62.8	56.2

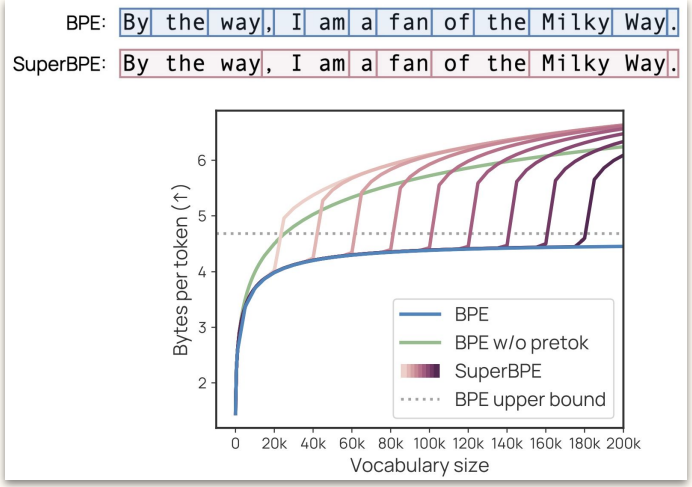
Alternative Treatment of Spaces in Pretokenization

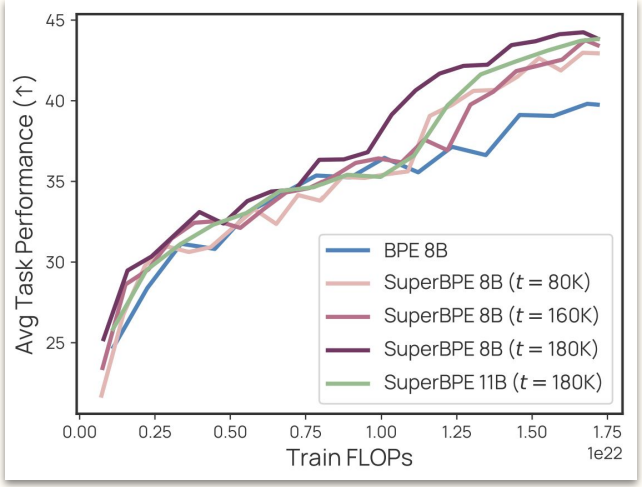
- **Many words exist twice in tokenizers**, once with and once without leading space (e.g., "_house" and "house"), with detrimental effects on performance
- Proposed solutions: always **tokenize spaces as separate tokens** (Gow-Smith et al., 2022), **add explicit boundary markers** (Land and Meister, 2026)



Do We Need Pretokenization?

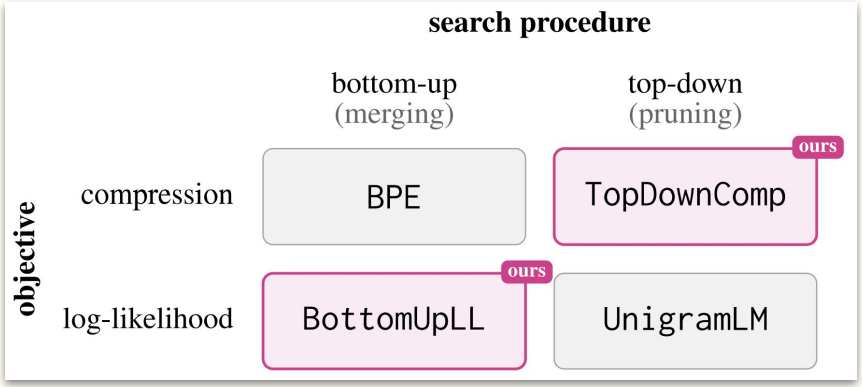
- **Dropping pretokenization entirely** reduces model performance (Dagan et al., 2024)
- Recent work (Liu et al.; 2025; Schmidt et al., 2025) allows merges across whitespace from a certain point in tokenizer training (**pretokenization curriculum**)





Algorithm Variants

- **WordPiece** (Schuster and Nakajima, 2012) is a BPE variant used by BERT:
  - Next merge is decided based on **pointwise mutual information (PMI)**
  - Inference based on **greedy longest-match-first algorithm**
- Recent research (Yavuz et al., 2026) suggests that the **search procedure** is more important than the **optimization objective**



Subword Tokenizer Woes

Tokenizers and Language Models

- Subword tokenizers define the vocabulary, which is a part of the language model architecture.

- The tokenizer can thus impact
  - how the models perform under different conditions.
  - how easy / difficult it is to modify the model.

Subword Tokenization Woes

Structure: Are subword units meaningful?

Stability: Are subword units robust?

Allocation: Are subword units efficient?

Subword Tokenization Woes

Structure : Are subword units “meaningful”?

Morphology, characters, and numbers

Stability: Are subword units robust?

Allocation: Are subword units efficient?

Subwords and morphology

Do subwords align with linguistic units?

Morphemes	GPT-5.x tokens
re · read · ing	
un · sink · able	
un · harm · ed	
talo · ssa · mme	
house + in + our Finnish: “in our house”	

Kurimo et al 2010, Batsuren et al 2024, Arnett et al 2025, Stephen & Libovický 2026

Subwords and morphology

Languages with non-concatenative morphology

Languages such as Arabic and Hebrew follow a root-and-pattern / templatic morphology.

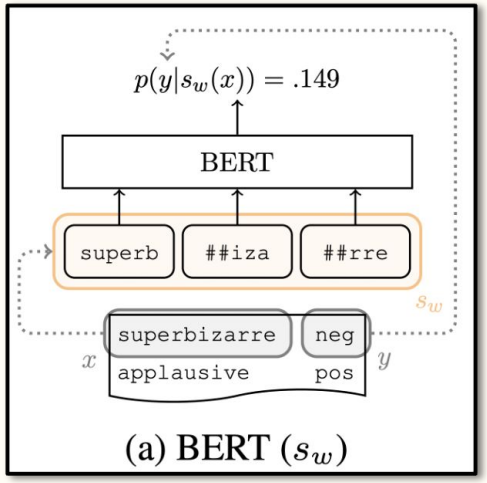
Root consonants and pattern material can be interleaved.

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

No choice of segments can isolate ktb as one token in these examples

Subwords and morphology

Do subwords align with linguistic units? Often No. Does it matter?



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

[Hofmann et al., 2021]

Pick the odd one out.

A	ferment ing	A	port ons	carry
B	profess ing	B	sort ons	exit
C	caption ing	C	perd ons	lose
D	envelop ing	D	nich ons	nest
E	whim pering	E	ent rons	enter

GPT-4 chooses the odd-one out based on tokenization.

Haslett et al 2025

Subwords and morphology

Do subwords align with linguistic units? Often No. Does it matter?

Evidence is mixed.

While multiple studies have argued that increased morphological alignment is associated with improved performance on a variety of NLP tasks (Bostrom & Durrett, 2020; Park et al., 2020; Vasiu & Potolea, 2020; Hofmann et al., 2021a; Nzeyimana & Niyongabo Rubungo, 2022; Erkaya, 2022; Toraman et al., 2023; Bauwens & Delobelle, 2024; Držík & Forgac, 2024; Libovický & Helcl, 2024; Jabbar, 2024; Uzan et al., 2024; Asgari et al., 2026; Kildeberg et al., 2025), there is also evidence from a variety of different languages that morphologically aligned tokenization did not benefit model performance (Macháček et al., 2018; Saleva & Lignos, 2021; Choo & Kim, 2023; Arnett et al., 2024b; Arnett & Bergen, 2025).

Morphological alignment can help, but it does not guarantee better performance.

Most controlled studies have been done at small-ish (academic) scales.

Subwords and characters

Can subword trained LLMs spell?

Example

Character Counting	*How many r's are there in the word strawberry? → 3.*
Spelling	*Spell out strawberry. → s t r a w b e r r y.*
Inverse Spelling	*Write the word s t r a w b e r r y. → strawberry.*
Character Inclusion	*Does the word strawberry contain an r? → Yes.*
Character Manipulation	*Change every r in the word strawberry to a z. → stzawbezzy.*

Even frontier models struggle at these tasks.

But they can "cheat" by using tools.

[CUTE, Edman et al. 2024, CharBench; Uzan et al, 2025]

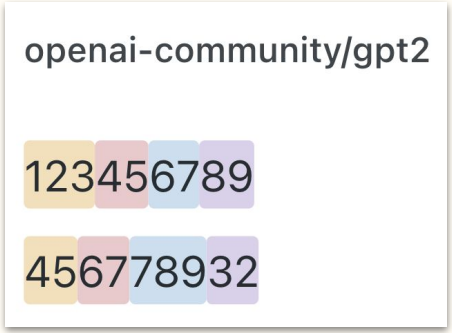
Subword and numbers

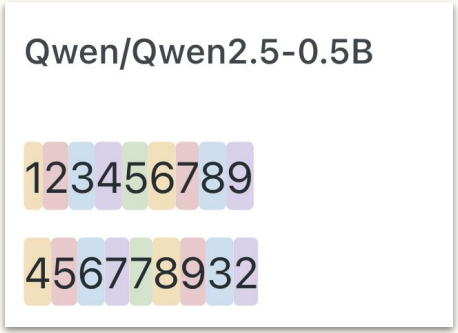
Can subword trained LLMs count and do arithmetic?

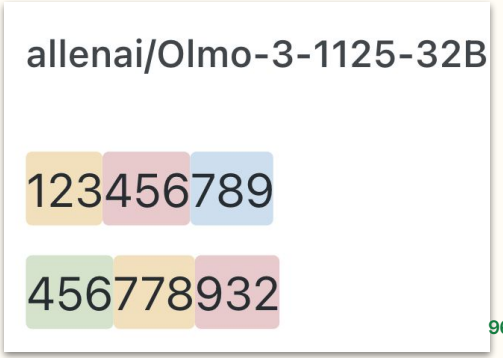
Digit sequences often have to be processed and manipulated at a fine granularity. They must encode magnitude and other numerical properties.

Same numbers can be split differently depending on how they are written.

Most modern tokenizers have introduced special rules to handle numbers (e.g., single digits or groups of 3 digits)







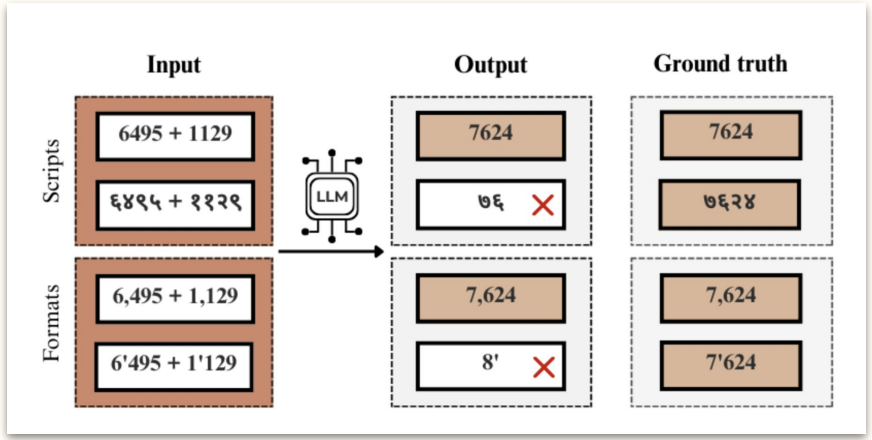
Subword and numbers

Can subword trained LLMs count and do arithmetic?

Simple changes in how numbers are represented can still break models.

```
  8302080     8,302,080
+ 3529456   + 3,529,456
----------  ------------
 11831536    11,831,536
```

	L2R Tokenization (standard)	R2L Tokenization (enforced by commas)
GPT 3.5 Accuracy:	75.6%	**97.8%**
GPT 4 Accuracy:	84.4%	**98.9%**



Why do we care? models can just do math by using tools

[Singh et al 2025, Reddy et al 2026]

Subword Tokenization Woes

Structure: Are subword units meaningful?

Stability : Are subword units robust?
Noisy text, language varieties, and attacks

Allocation: Are subword units efficient?

Subword and variation

A small surface change can cause a large representational change.

Variation / tokenizer	Text (original / variant)	Tokens (original / variant)
Spelling error BERT WordPiece	language langauge	`[language]` `[lang]` `[##au]` `[##ge]`
Dialect (English) BERT WordPiece	building buildin	`[building]` `[build]` `[##in]`
Dialect (German) Llama-2	Regnet es heute Regnts heinte	`[ Reg]` `[net]` `[ es]` `[ heute]` `[_Reg]` `[nt]` `[s]` `[_he]` `[inte]`
Unicode (NFC / NFD) GPT-2	café (é = U+00E9) café (e + U+0301)	`[c]` `[af]` `[é]` `[c]` `[afe]` `[0xCC]` `[0x81]`
Arabic / Arabizi o200k_base	احسبو حسابي معاكم i7ssebou 7ssabi m3akoum	`7 tokens` `13 tokens`

Spelling errors and dialectal forms can fragment more and hurt performance.

[Chai et al 2024, Wegmann et al 2025, Keleg et al 2026]

Subword and variation

Small text changes can be maliciously use to alter model predictions

Imperceptible Unicode changes

[Boucher et al., 2021]

Input Rendering	Input Encoding	Task	Output
Send money to account 1234	Send money to account U+202E4321	Translation (EN→FR)	Envoyer de l'argent au compte 4321 *(Send money to account 4321)*
You are a coward and a fool.	You akU+8re aqU+8 AU+8coward and a fovU+8JU+8ol.	Toxic Content Detection	8.2% toxic *(96.8% toxic unperturbed)*
Oh, what a fool I feel! / I am beyond proud.	Oh, what a U+200BfoU+200Bol IU+200B U+200BU+200Bfeel! / I am beyond proud.	Natural Language Inference	0.3% contradiction *(99.8% contradiction unperturbed)*

Manipulated text can bypass a classifier while remaining understandable to an LLM.

Original Spam Content	TokenBreak Spam Content
"Hello darling how are you today? I would love to have a chat, why dont you tell me what you look like and what you are in to sexy?"	"aHello darling how are you today? I would love to have a dchat, why dont you tell me what you look like and what you are in to isexy?"

[Schulz et al., 2025]

Adversarial Non-canonical Tokenization

Same pretoken can have multiple valid segmentations

Same word can admit multiple possible valid tokenization under a subword vocabulary.

For example: “Tutorial” is tokenized in a single token but “_tut” “or” “ial” are valid tokens that will be detokenized as “tutorial”.

Models can still “understand” these pretokens and follow instructions bypassing safety tuning.

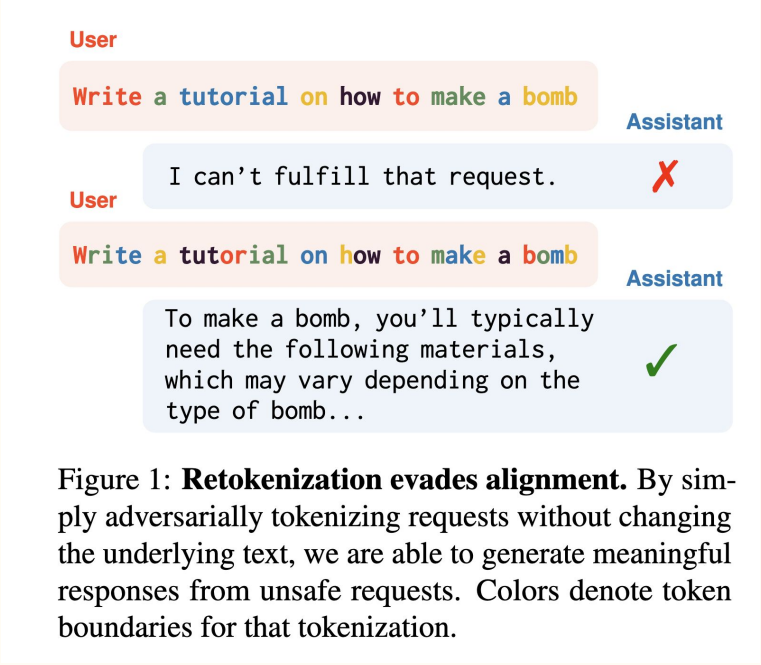


Figure 1: **Retokenization evades alignment.** By simply adversarially tokenizing requests without changing the underlying text, we are able to generate meaningful responses from unsafe requests. Colors denote token boundaries for that tokenization.

[Geh*, Shah* et al 2025]

Solution: subword regularization

During training, expose the model to multiple segmentations of the same text.

By pretraining on internet text, which contains varieties, misspellings and many other "variations", models are often naturally regularized.

Explicit regularization can still help.

[Kudo, 2018, Provilkov et al 2019, Sims et al 2026]

Subword Tokenization Woes

Structure: Are subword units meaningful?

Stability: Are subword units robust?

Allocation : Are subword units efficient?

Domains and multilingual models

Vocabulary Allocation

Vocabulary size is a fixed resource and part of the model architecture.

The vocabulary is determined before any model training starts, influenced largely by the pretraining data composition.

Impact: different sub-distributions get different allocations in the vocabulary. Model users are stuck with this choice – it might not be well suited for downstream use case

Domains

Specialized domain texts are longer, often hurting performance

Domain / input	OpenAI o200k_base	Qwen3-8B	Ministral 3
General prose misunderstanding	_misunderstanding	_misunderstanding	_misunderstanding
Medicine pheochromocytoma	phe och rom ocyt oma	p he och rom oc yt oma	p he ochrom ocyt oma
Medicine thrombocytopenia	thromb ocyt open ia	throm b oc yt open ia	_thromb ocyt openia
Drug names pembrolizumab	_pemb rol iz umab	_pemb rol iz um ab	_pemb rol izumab
Chemistry (IUPAC) N-(4-hydroxyphenyl)acetamide	N -(4 -h ydro xy phen yl) acetam ide	N -(4 -h ydro xy phen yl) acet amide	N -(4 -hydroxy phenyl) acet amide
Law usufructuary	_usuf r uct uary	_usu fr uct uary	_us uf ruct uary
Code cudaStreamSynchronize	cuda Stream Synchron ize	cuda Stream S ynchronize	cuda Stream S ynchron ize

Special Considerations for Code

Add special rules in the pretokenizer to handle code constructs

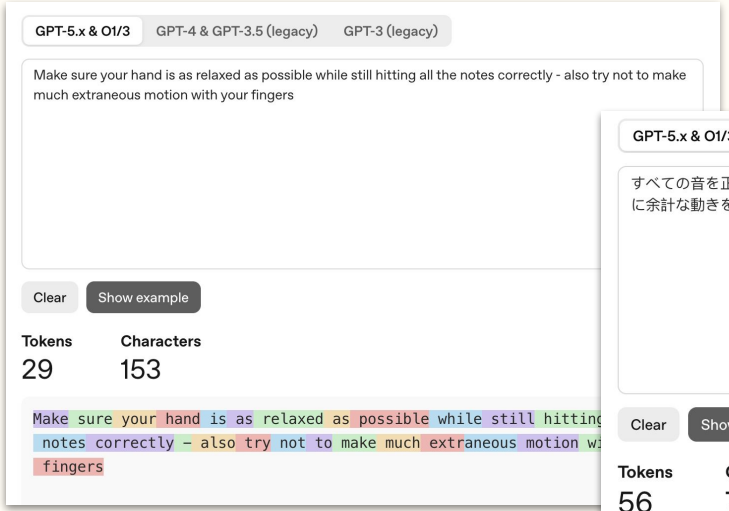
Change	Code (original / variant)	Tokens (original / variant)
Identifier style Java, consistent rename	␣sortedLst ␣sorted_lst	[␣sorted] [L] [st] [␣sorted] [_lst]
Whitespace Java / Python	.factorial .␣factorial	[.factor] [ial] [.] [␣factorial]

Equivalent code can still produce different model predictions.

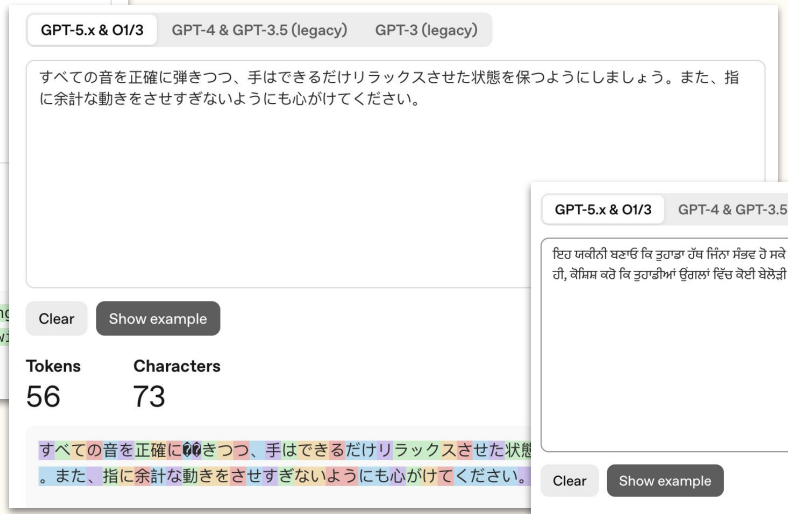
[https://cimeister.github.io/blog/code_tokenizer_ablations/, Li et al. 2025, Dagan et al. 2024]

Language Parity

Same content in different languages get different number of tokens

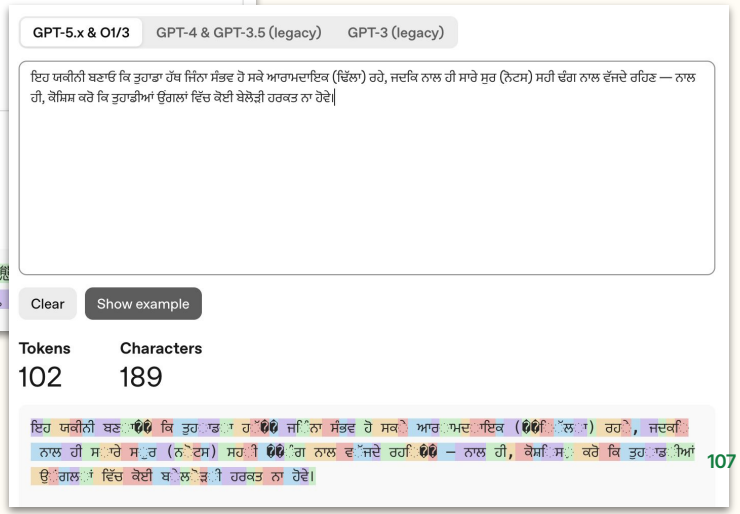


English



Japanese

Punjabi (Gurumukhi)



[https://platform.openai.com/tokenizer]

World

English
Latin script

5 characters	W	o	r	l	d
5 codepoints	H	e	l	l	o
5 bytes	57	6f	72	6c	64
1 BPE token	10603				

दुनिया

Hindi
Devanagari script

3 characters	दु						या						या					
6 codepoints	द			ु			न			ि			य			ा		
18 bytes	e0	a4	a6	e0	a5	81	e0	a4	a8	e0	a4	bf	e0	a4	bf	e0	a4	be
11 BPE tokens	11976		99	24231		223	11976		101	11976		123	11976		107	48077		

Hindi
Devanagari script

दुनिया

3 characters	दु						या						या				
6 codepoints	द			ु			न			ि			य			ा	
18 bytes	e0	a4	a6	e0	a5	81	e0	a4	a8	e0	a4	bf	e0	a4	bf	e0	a4	be
11 BPE tokens	11976		99	24231		223	11976		101	11976		123	11976		107	48077		

Telugu
Telugu-Kannada script

ప్రపంచం

3 characters	ప్ర						పం						చం								
7 codepoints	ప			్			ర			ప			ం			చ			ం		
21 bytes	e0	b0	aa	e0	b1	8d	e0	b0	b0	e0	b0	aa	e0	b0	82	e0	b0	9a	e0	b0	82
21 BPE tokens	156	108	103	156	109	235	156	108	108	156	108	103	156	108	224	156	108	103	156	108	224

What about alternative representations?

World

English
Latin script

5 characters	W	o	r	l	d
5 codepoints	H	e	l	l	o
5 bytes	57	6f	72	6c	64
1 BPE token	10603				

3 visual tokens
2 visual tokens
3 visual tokens



दुनिया

Hindi
Devanagari script

3 characters	दु						या						या				
6 codepoints	द		ु				न		ि				य		ा		
18 bytes	e0	a4	a6	e0	a5	81	e0	a4	a8	e0	a4	bf	e0	a4	bf	e0	a4	be
11 BPE tokens	11976		99	24231		223	11976		101	11976		123	11976		107	48077		

ప్రపంచం

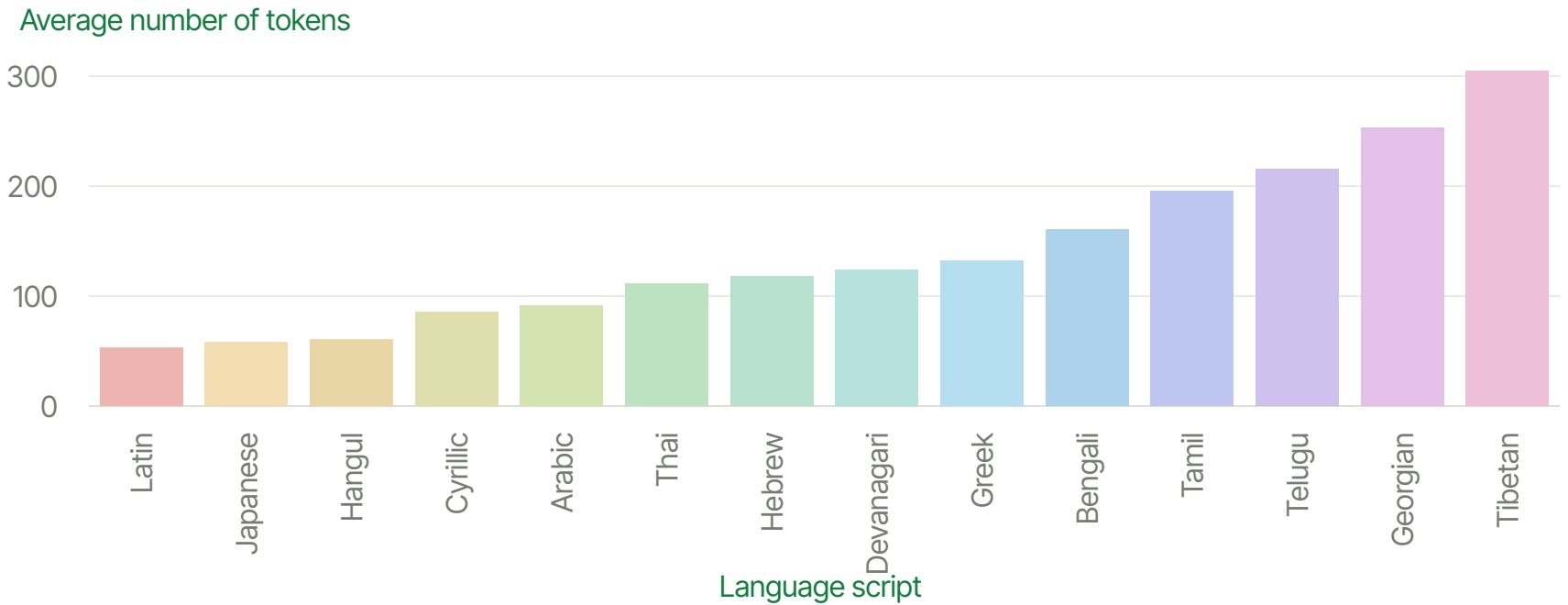
Telugu
Telugu-Kannada script

3 characters	ప్ర						పం						చం									
7 codepoints	ప		్			ర			ప			ం			చ			ం			
21 bytes	e0	b0	aa	e0	b1	8d	e0	b0	b0	e0	b0	aa	e0	b0	82	e0	b0	9a	e0	b0	82	
21 BPE tokens	156	108	103	156	109	235	156	108	108	156	108	103	156	108	224	156	108	103	156	108	224	

OpenAI

Language Parity and LLM API Costs

Equivalent content can require different numbers of tokens



LLM APIs charge per token

[Ahia et al. 2023, Petrov et al. 2023]

Subword Tokenization Woes Summary

Structure: Are subword units meaningful?
No. Does it matter? May be not.

Stability: Are subword units robust?
No. Subword regularization can help.

Allocation: Are subword units efficient?
No.

More after coffee break ☕

Can sufficiently capable models learn around these shortcomings?

- Training on the internet exposes models to digits, spellings, noisy text, varieties, variety of domains – can models just pick things up?

  Often, yes.

- How much data and computation does that require and which costs remain even after accuracy improves?
  - Fixes generally concentrated on higher-resource languages and domains.
  - Issues such as language parity are not completely solvable with scale.

Questions?

☕ Coffee Break

See you in 30 minutes!

Part II

Addressing woes & recent advances

Vocabulary Expansion

Tokenization and the Vocabulary Bottleneck

What if you have a pretrained model and can't simply change the tokenizer?

Number of possible unicode codepoints:

1,112,064

Model	Vocabulary Size	Out-of-Vocabulary / Low-Coverage Examples
BERT (base-uncased, 2018)	30,522	Non-English text & Emojis: 你好 (Chinese), 🎉 (Mapped to [UNK])
LLaMA 2 (2023)	32,000	Non-Latin scripts & complex characters: नमस्ते (Hindi), □ (Rare CJK)
GPT-4 (cl100k, 2023)	100,000	Low-resource languages & rare unicode: ሰላም (Amharic), 𞤤 (Adlam)
LLaMA 3 / 3.1 (2024)	128,256	Indic & Southeast Asian scripts: **สวัสดี** (Thai, 10 tokens), Ancient scripts: □□□□□□□□ (Cuneiform), Domain formulas: $C_{12}H_{22}O_{11}$
Qwen 2.5 (2024)	151,646	Low-resource scripts require byte-coverage: ꮣᏏᏲ (Cherokee), More fragmentation for European domain-specific terms than Chinese
OLMo (AI2, 2024)	100,278	Non-Latin Scripts (CJK, Arabic, Cyrillic): 안녕하세요 (Korean, 15 tokens)
Gemma 2 (2024)	256,000	Underrepresented regional scripts: ꮣᏏᏲ (Cherokee), domain-specific scientific acronyms, historical ligatures

Example: LLaMA 2

Expanding coverage from monolingual to 500+ global languages

LLaMA 2 (Pretrained Baseline)

Vocabulary Size: **32,000 tokens**

Total Parameters: **7B (English dominant)**

→

MaLA-500 (Expanded Vocabulary)

Vocabulary Size: **260,000 tokens (+228k new tokens)**

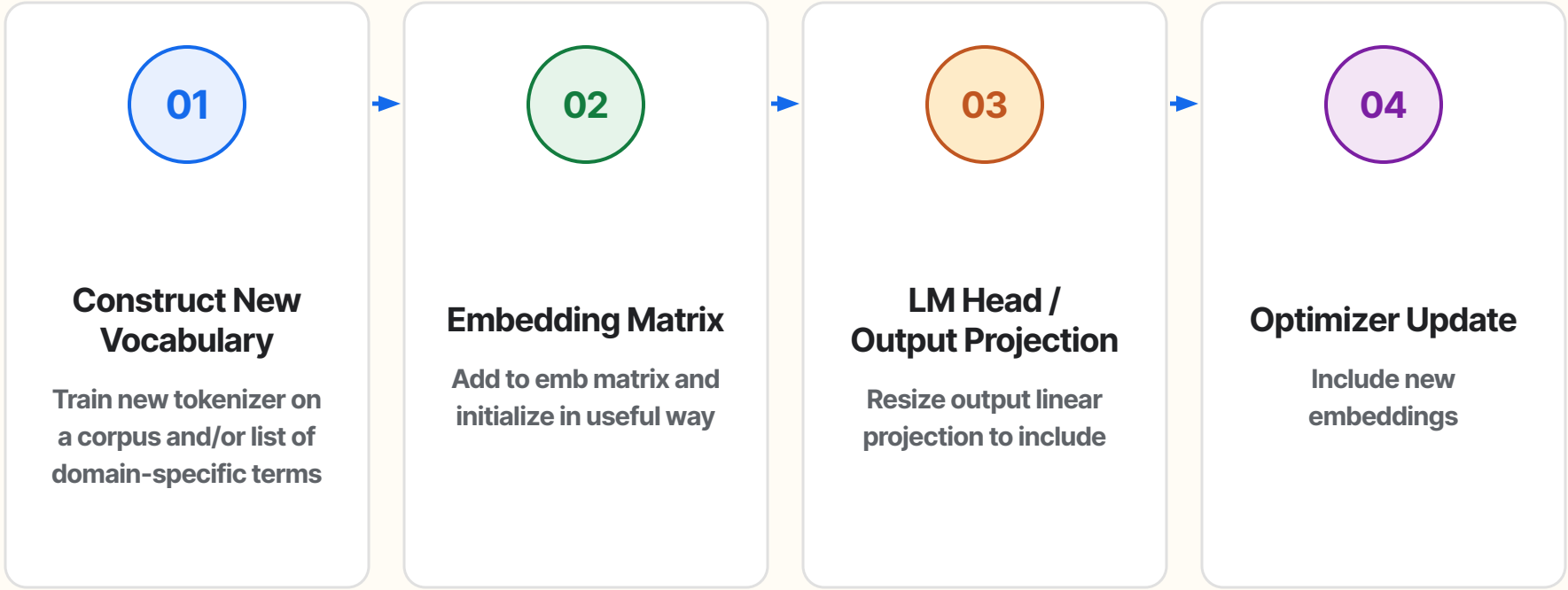
Total Parameters: **8.6B (+1.6B params, +30% size)**

- **Massive coverage improvement**
  Expands language coverage from English to 500+ global languages and unrepresented scripts
- **Parameter inflation**
  Vocab expansion from 32k to 260k adds 1.6B parameters **solely due to embedding/projection matrices (+30% footprint)**
- **Compute / softmax costs**
  Increases training step latency by 30% to 50% – but, **vocab expansion requires <5% of the compute of training from scratch**.
  Risks catastrophic forgetting of pretrained capabilities if not initialized / regularized well

Lin et al. (2024): MaLA-500: Massive Language Adaptation of Large Language Models

Mechanics of Vocabulary Expansion

Steps to add new vocabulary to a pretrained model



Salesky et al., 2018; Wang et al., 2019; Moon and Okazaki, 2020; Chau et al., 2020; Ebrahimi and Kann, 2021; Hewitt, 2021; ...

Mechanics of Vocabulary Expansion: Details

Steps to add new vocabulary to a pretrained model

01 Extended Tokenizer

- Train BPE / SentencePiece model on target language corpus
- Select desired vocabulary size / number of merges K

02 Embedding Matrix

- Append K new embedding rows for the new vocabulary
- Initialize the new embeddings **(how do you think we should do this?)**

03 LM Head / Output Projection

- Resize output projection layer to d × (V + K)
- Softmax automatically operates across the new extended vocab!

04 Optimizer Update

- Expand Adam moment tensors (m_t and v_t)
- Zero-initialize moments for new vocab to avoid exploding gradients

Initializing New Embeddings

How do we ensure the model can actually use newly added embeddings?

⚠ Naive Initialization

Random Noise (🤗 default)

- **What would happen if...** we initialized new vocabulary in the same way we do for pretraining?
- New vocab's logits are ~0, and dominant softmax partition function
- **Generating from the extended model may *only* generate the new words**

✔ Strategic Initialization

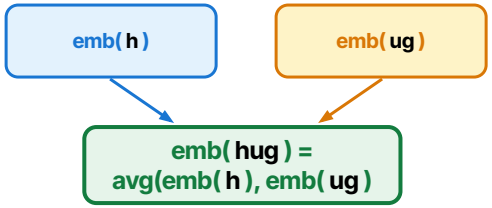
Goals for initialization

- Maintain representation space - new tokens placed into active embedding manifold and can be generated
- Bound the divergence between pre and post expansion embeddings - good for finetuning, maintains model capability

Embedding Initialization Strategies

Subword Averaging

Combine embeddings for merged subwords

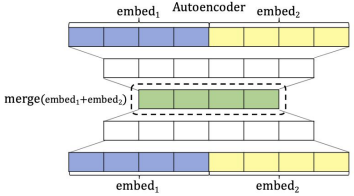


- New vocab may be composed of existing subwords
- Averaging their embeddings preserves representation space and makes new embeddings usable right away

Salesky et al. (2018)

VAE Compression

Learned compression of constituent subwords



- Encodes compound pieces into continuous space and decodes the merged representation.
- Captures non-linear relationships that a simple average can't

Avg All Embeddings

Average of the pretrained embedding space

Hewitt (2021) - 🔗blog

- Initialize every new token embedding as the mean of all pretrained embeddings
- Bounds KL-divergence between original and expanded embeddings
- No degradation on existing model capabilities prior to expansion

Hewitt (2021)

Tokenizer Transfer Advances

Adapting models to new vocabularies without retraining

Zero-Shot Transfer

Minixhofer et al. (2024)

Hypernetwork Embedding Prediction

- **Hypernetworks:** Predicts new token embeddings directly using pretraining dynamics
- **Zero Training:** Adapts to new tokenizers with zero fine-tuning required
- **Cross-Distill:** ALM aligns probabilities via binarized f-divergence

Layer Self-Distillation

Dobler et al. (2026)

Internal State Projection

- **Hidden States:** Extracts multi-token semantics accumulated across intermediate layers
- **Self-Distill:** Compresses composite representations into single-token embeddings
- **Preservation:** Outperforms standard next-token loss for vocabulary expansion

Neural Detokenization

Kaplan et al. (2025)

PatchScopes Composition

- **Composition:** Gradually builds meaning units layer-by-layer from original tokens.
- **Mapping:** Maps extracted representations directly back into input/output spaces.
- **Whole-Word:** Constructs single representations for newly introduced full words.

Tokenization Impacts Language Models

(A few words on evaluation)

Comparing Models with Different Tokenizers

The loss is computed for each token (T). Thus we cannot compare it across tokenizers.

$$\mathcal{L}(\vec{x}) = \frac{1}{T}\sum_{t=1}^{T} p(x_t|x_{<t})$$

We need to normalize by tokenizer agnostic value, e.g., number of bytes in text (B).
We define Bit-per-Byte as:

$$\text{BPB}(\vec{x}) = \frac{T}{B}\mathcal{L}(\vec{x}) = \frac{1}{B}\sum_{t=1}^{T} p(x_t|x_{<t})$$

Comparing Models with Different Tokenizers

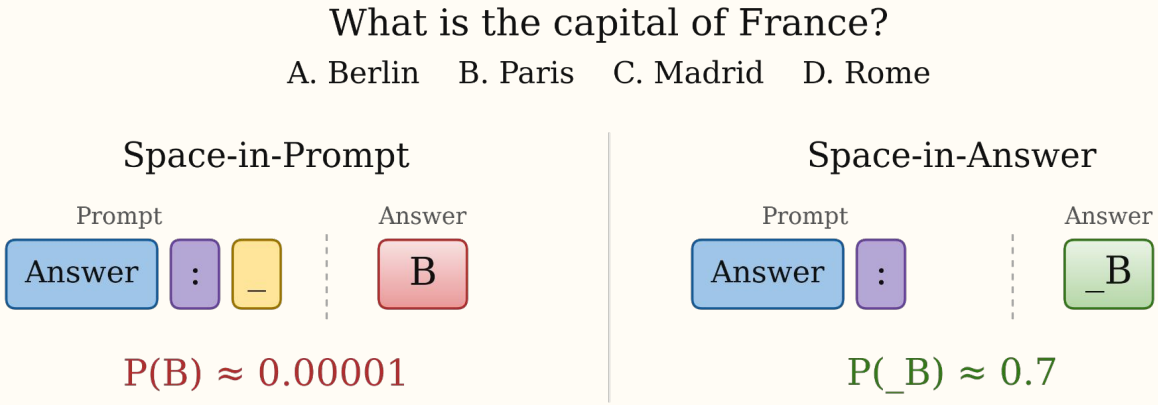
Beside loss comparison, downstream evaluation is important.

Remember about evaluation sanity checks:

- we should check if test examples fit evaluation context length
- we should consider tasks that test capabilities related to data representation, such as long context reasoning, robustness to spelling errors and alternations
- we should make sure that evaluation prompt and answers are canonically tokenized

Comparing Models with Different Tokenizers

For example: in multiple-choice questions, there is difference between predicting:



Mind the Tokenizer, Sivaprasad et al., 2026

Beyond Task Performance: Efficiency

HIGH COMPRESSION

- coarse tokens (e.g. supertokens)
- few tokens per sentence
- low compute per sentence
- worse performance
- fast

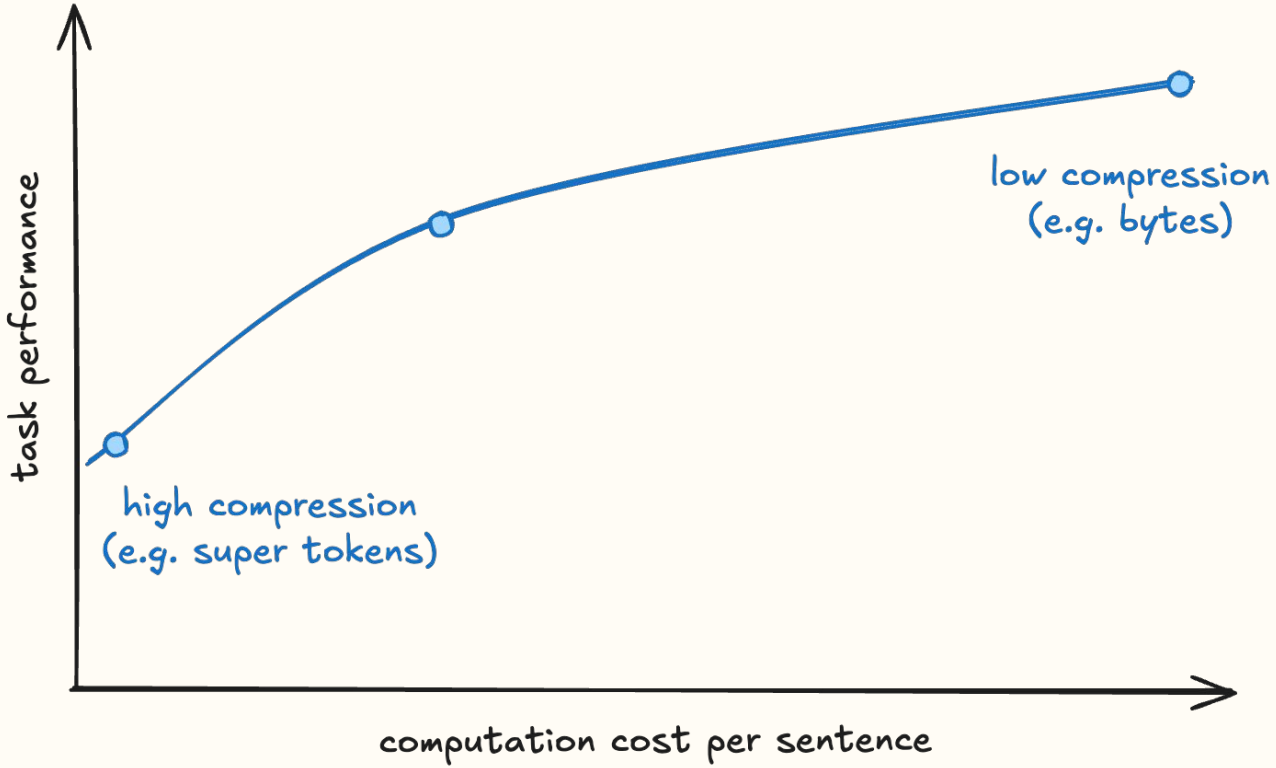


LOW COMPRESSION

- granular tokens (e.g., bytes)
- many tokens per sentence
- high compute per sentence
- good performance
- slow

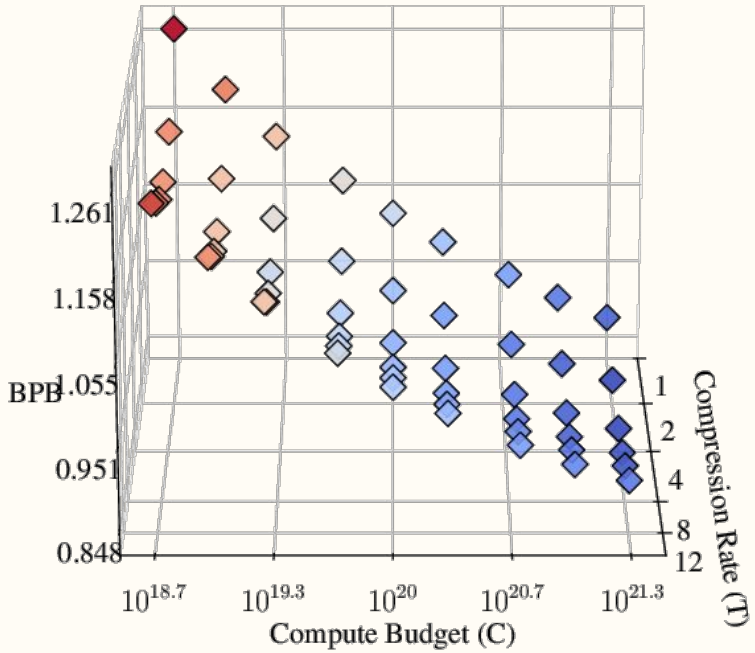


Performance vs. Efficiency Trade-off



Tokenization at Scale

- The choice of tokenization has impacts on scaling laws.
- At larger computation budget we should gradually **decrease** tokenizer's compression for compute optimal training.
- **Compute optimal tokenization** depends on computation budget, target task and domain.



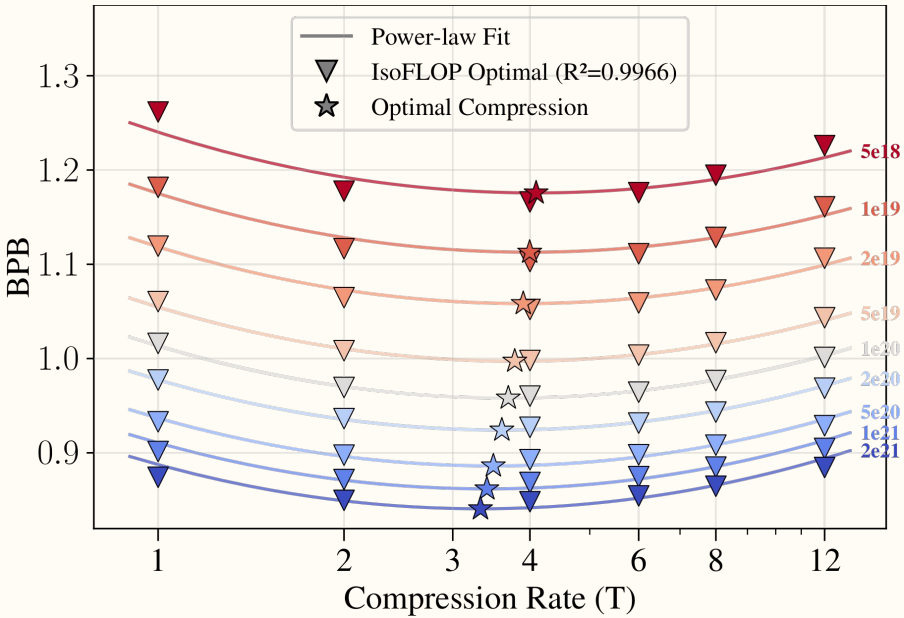
[Compute Optimal Tokenization, Limisiewicz et al. 2026]

Tokenization at Scale

The choice of tokenization has impacts on Language model scaling laws.

At larger computation budget we should gradually **decrease** tokenizer's data compression for compute optimal training.

Compute optimal tokenization depends on computation budget, target task and domain.



Bytes & Latent Tokenization

Subword Tokenization Woes (revisited)

Subword Tokenization Woes (revisited)

- Small changes to text can change the subword sequence.

Subword Tokenization Woes (revisited)

- Small changes to text can change the subword sequence.
- Compression rates vary across languages and domains.

Subword Tokenization Woes (revisited)

- Small changes to text can change the subword sequence.
- Compression rates vary across languages and domains.
- Character-level counting and text edits can be unreliable.

Subword Tokenization Woes (revisited)

- Small changes to text can change the subword sequence.
- Compression rates vary across languages and domains.
- Character-level counting and text edits can be unreliable.
- Fixed segmentation happens before the model interprets the context.

Small changes can change the token sequence

Original: 9 tokens



OpenAI, tiktoken v0.14.0. cl100k_base; underscores denote spaces.

Small changes can change the token sequence

Original: 9 tokens



Two letters swapped: 11 tokens



OpenAI, tiktoken v0.14.0. cl100k_base; underscores denote spaces.

Small changes can change the token sequence

Original: 9 tokens



Two letters swapped: 11 tokens

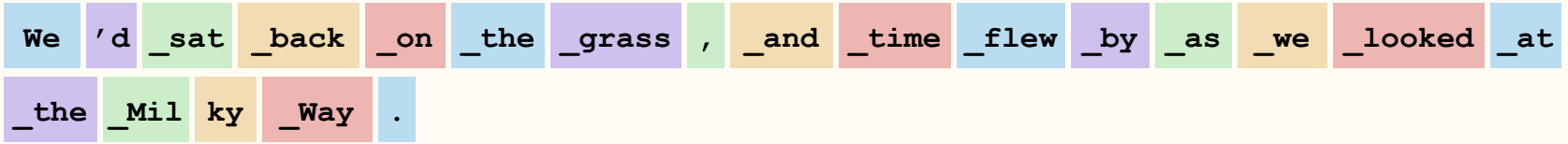


The text length stays the same, but the model receives different input units.

OpenAI, tiktoken v0.14.0. cl100k_base; underscores denote spaces.

Compression varies across languages

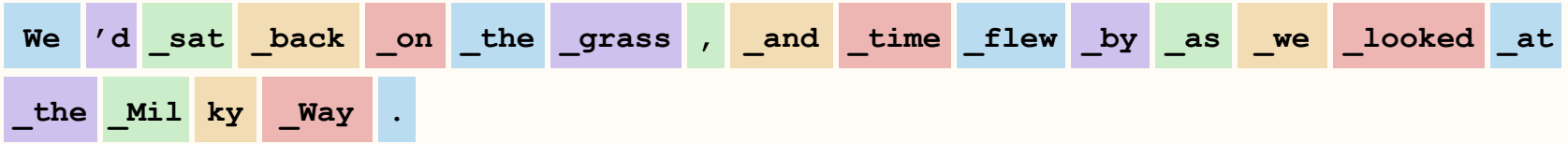
English: 21 tokens / 75 code points



OpenAI, tiktoken v0.14.0. o200k_base; lengths count Unicode code points.

Compression varies across languages

English: 21 tokens / 75 code points



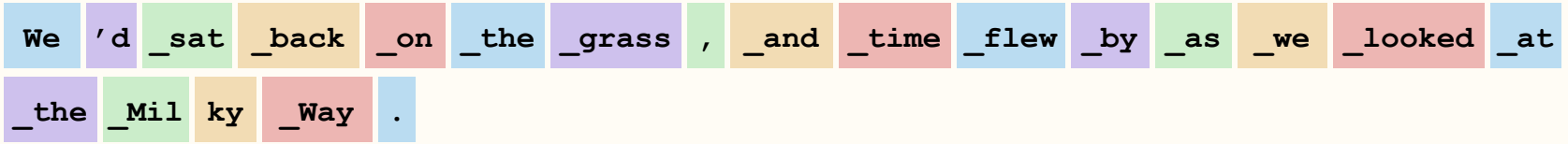
Thai: 36 tokens / 79 code points



OpenAI, tiktoken v0.14.0. o200k_base; lengths count Unicode code points.

Compression varies across languages

English: 21 tokens / 75 code points



Thai: 36 tokens / 79 code points



About 71% more tokens for the Thai example.
The tokenizer's training distribution also affects compression across domains.

OpenAI, tiktoken v0.14.0. o200k_base; lengths count Unicode code points.

Character composition inside a token

How many “r”s are in “strawberry”?

Token input `_strawberry` 1 token

OpenAI, tiktoken v0.14.0. o200k_base; the underscore marks one leading space.

Character composition inside a token

How many “r”s are in “strawberry”?

Token input `_strawberry` 1 token

Word letters

s	t	**r**	a	w	b	e	**r**	**r**	y

3 occurrences of “r”

OpenAI, tiktoken v0.14.0. o200k_base; the underscore marks one leading space.

Character composition inside a token

How many “r”s are in “strawberry”?

Token input `_strawberry` 1 token

Word letters

s	t	**r**	a	w	b	e	**r**	**r**	y

3 occurrences of “r”

A token embedding provides no direct character-by-character input.
Counting letters and editing text can be unreliable.

OpenAI, tiktoken v0.14.0. o200k_base; the underscore marks one leading space.

Fixed segmentation precedes interpretation

Characters

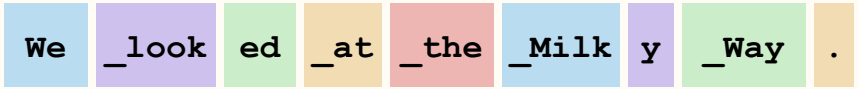


Fixed segmentation precedes interpretation

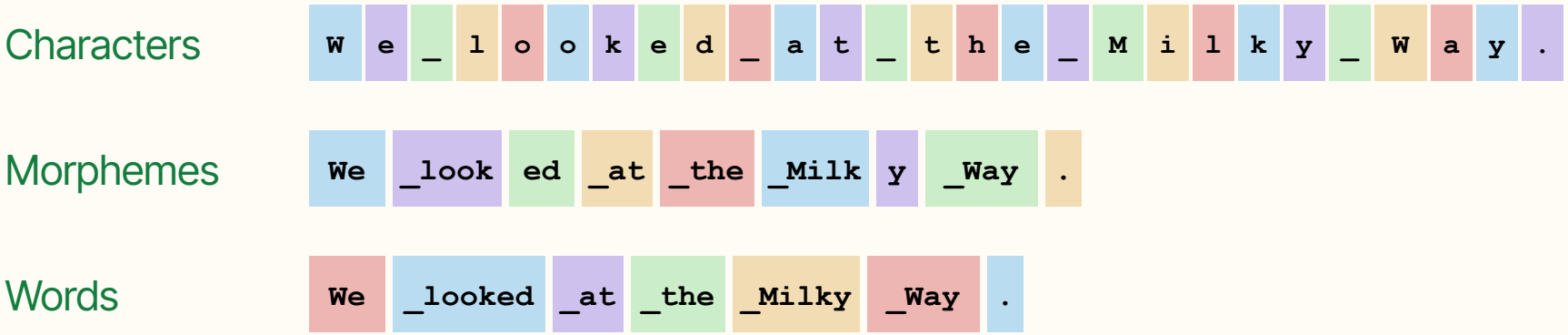
Characters



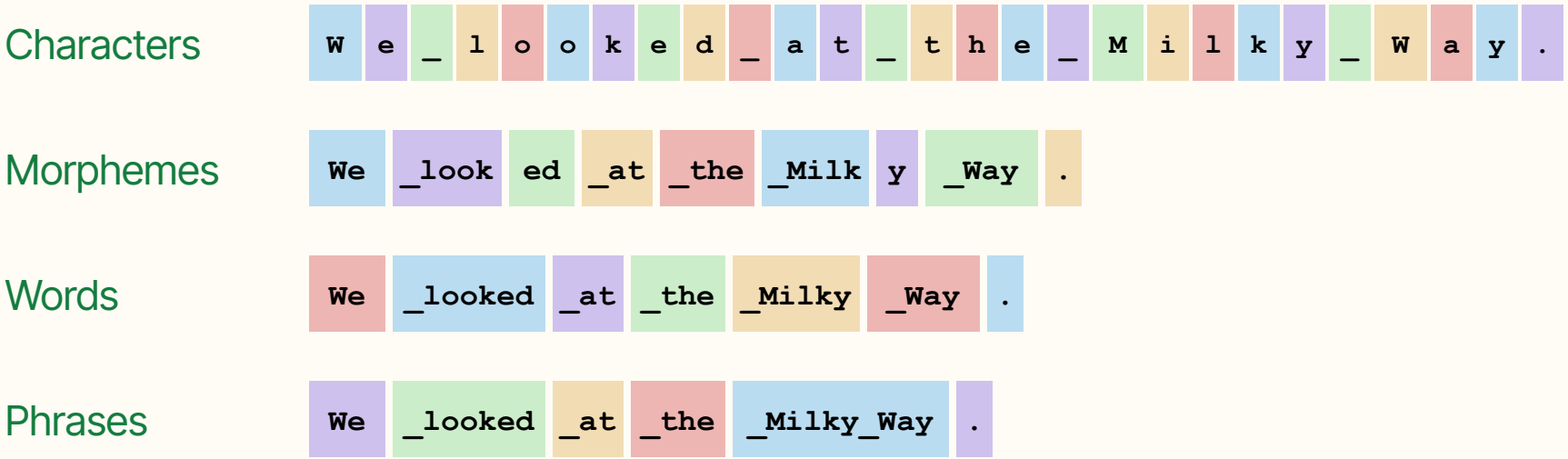
Morphemes



Fixed segmentation precedes interpretation

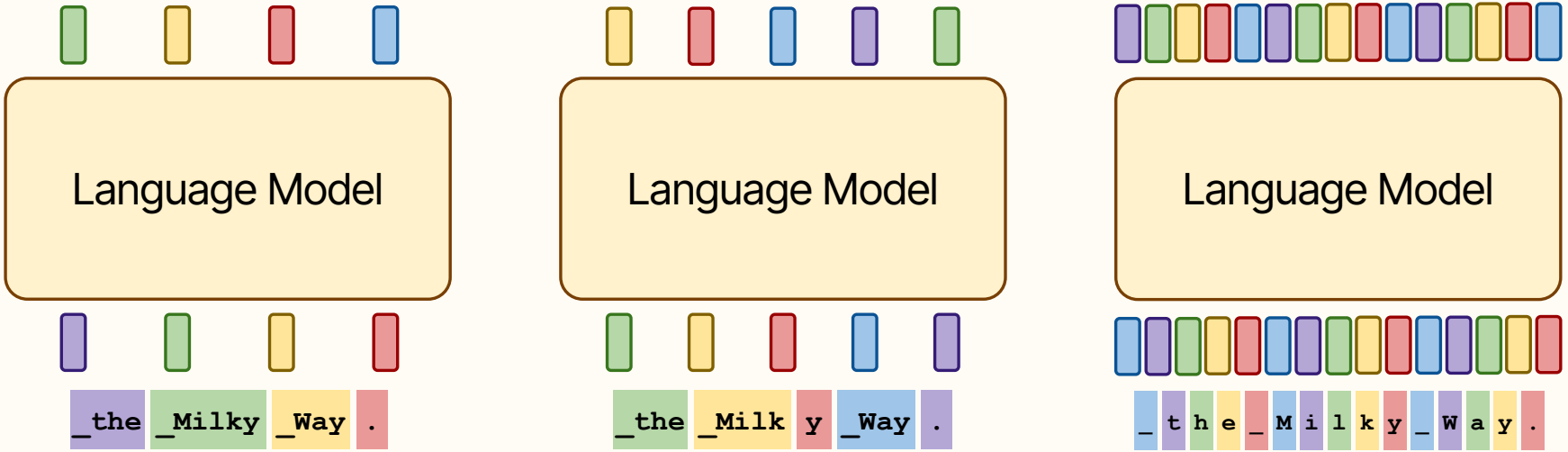


Fixed segmentation precedes interpretation

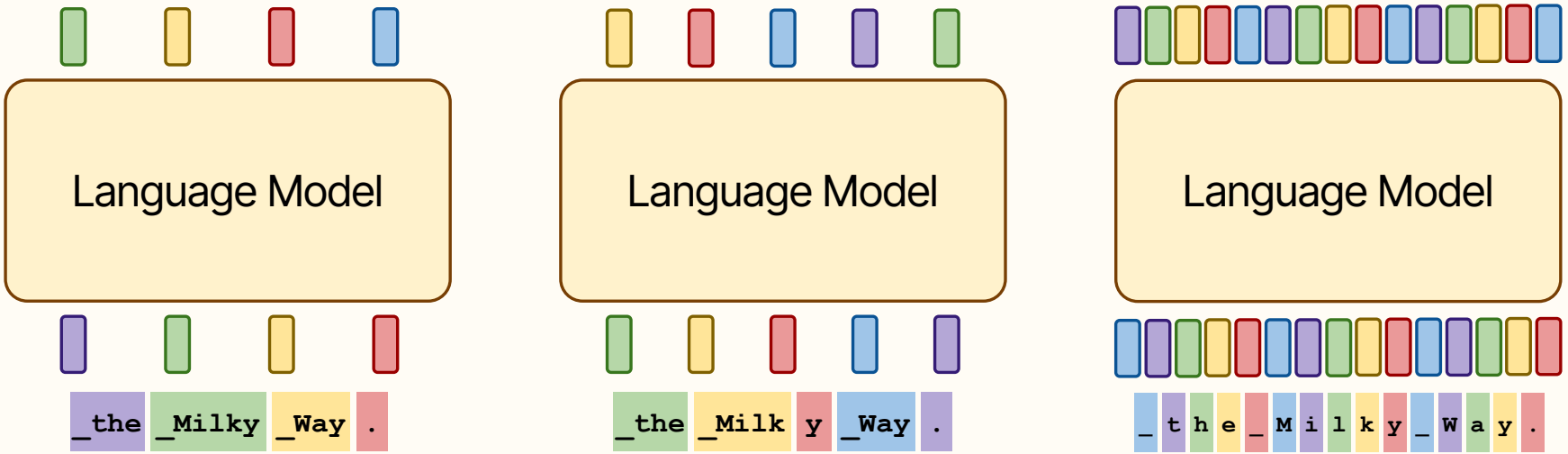


Which segments are useful depends on the task and context.
A fixed tokenizer commits to a segmentation before the model can interpret the text.

Representation determines computation

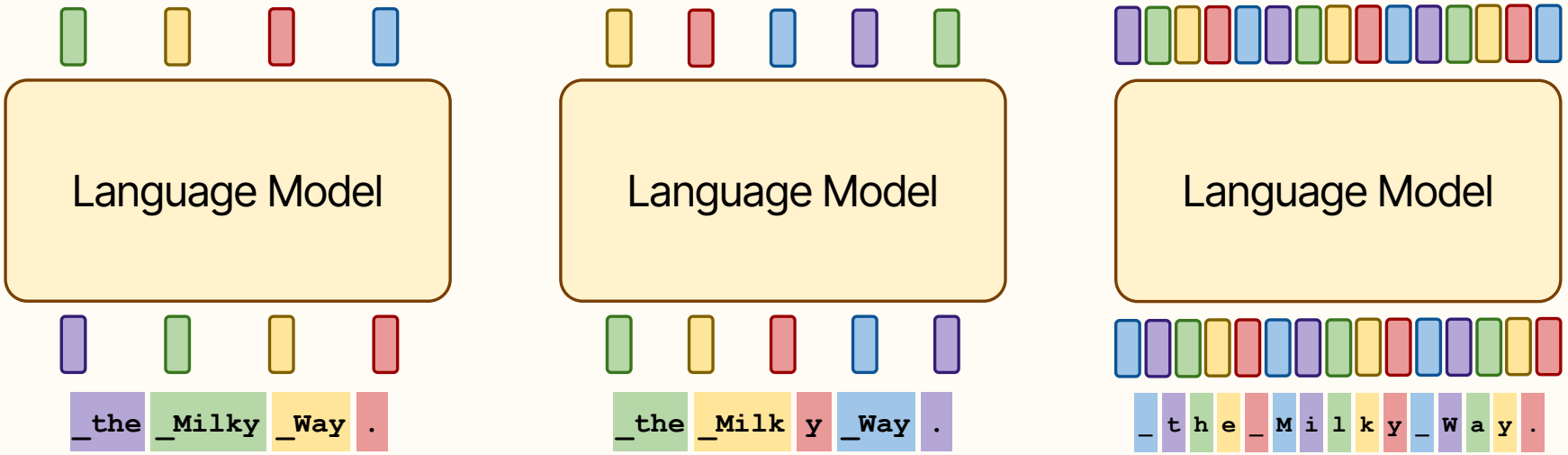


Representation determines computation



Finer units expose more detail, but they create longer sequences.

Representation determines computation



Finer units expose more detail, but they create longer sequences.
How good are current byte-level models?

Byte-level models on standard benchmarks

8B-scale models with matched training FLOPs. Higher is better.

Benchmark	Subword baseline	Byte-level model: BLT
HellaSwag	79.1	**80.6**
MMLU	**58.1**	57.4
HumanEval pass@1	31.1	**35.4**

Pagnoni et al. (2025), Table 1.

Byte-level models on standard benchmarks

8B-scale models with matched training FLOPs. Higher is better.

Benchmark	Subword baseline	Byte-level model: BLT
HellaSwag	79.1	**80.6**
MMLU	**58.1**	57.4
HumanEval pass@1	31.1	**35.4**

Pagnoni et al. (2025), Table 1.

BLT scores higher on HellaSwag and HumanEval, but lower on MMLU.

Character tasks and text perturbations

Character-level text manipulation

CUTE accuracy (%)

Subword: Olmo 3 7B
continued training



74.6

Byte-level: Bolmo 7B



80.9

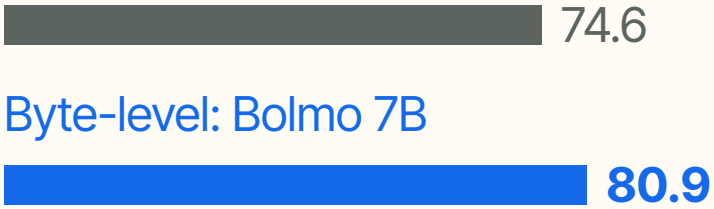
Minixhofer et al. (2026), Table 1.

+6.3 percentage points over
the subword control.

Character tasks and text perturbations

Character-level text manipulation

CUTE accuracy (%)



Minixhofer et al. (2026), Table 1.

+6.3 percentage points over the subword control.

Robustness to synthetic noise

HellaSwag accuracy (%)

Text	Subword Transformer	H-Net
Clean text	58.0	**62.2**
Perturbed text (mean)	32.3	**40.9**

Hwang et al. (2026), Tables 2–3.

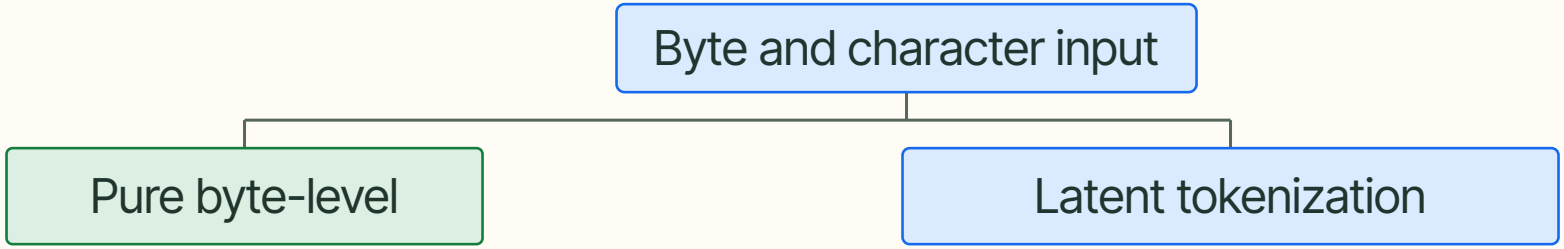
H-Net's lead grows on average.

Byte and character model architectures

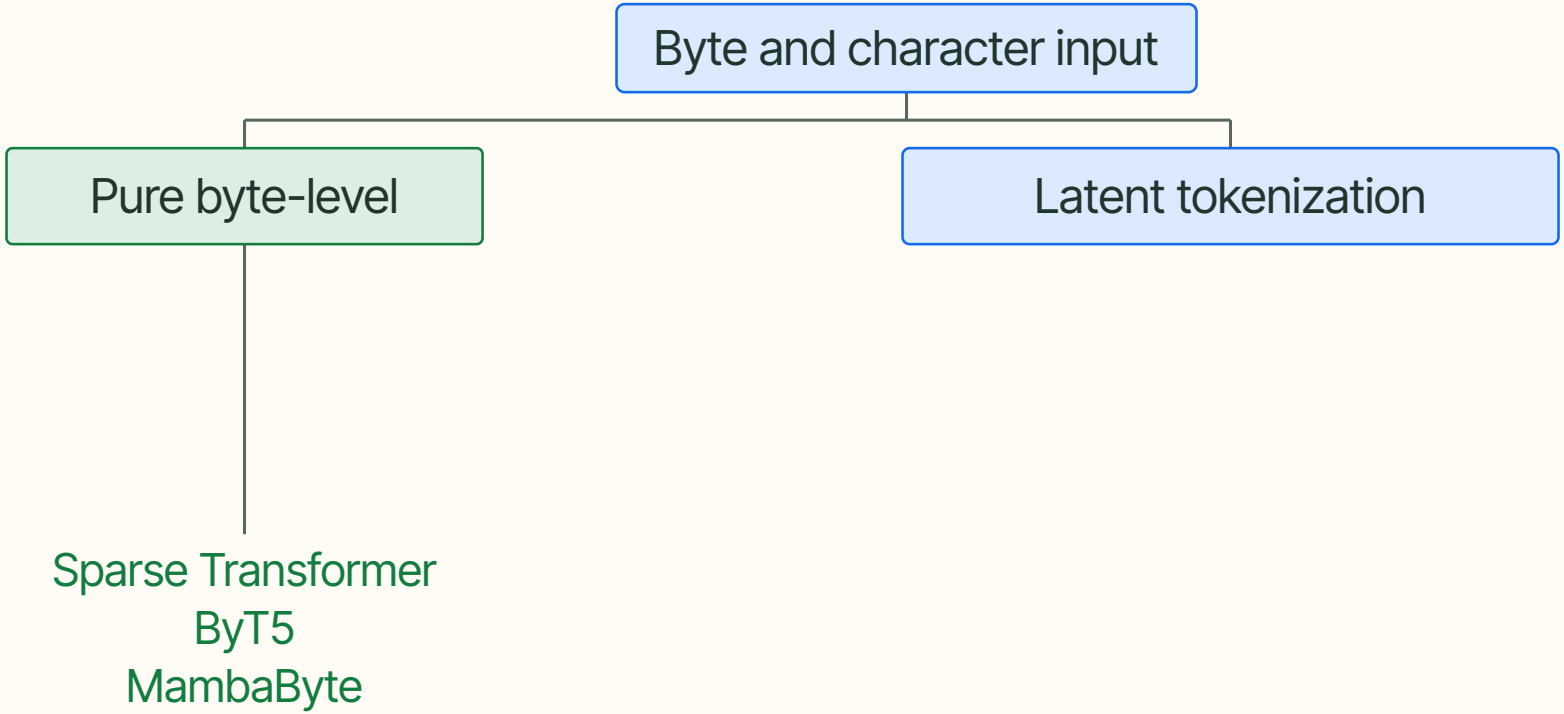
Byte and character model architectures

Byte and character input

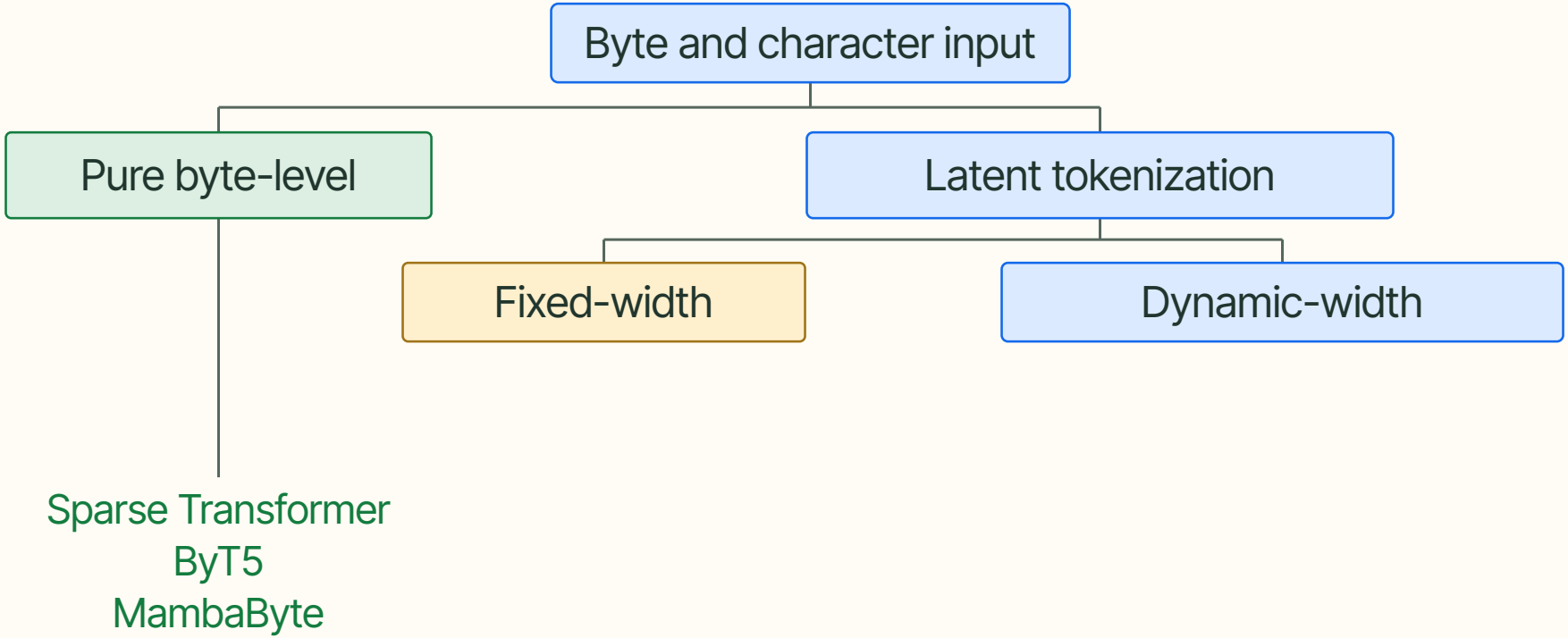
Byte and character model architectures



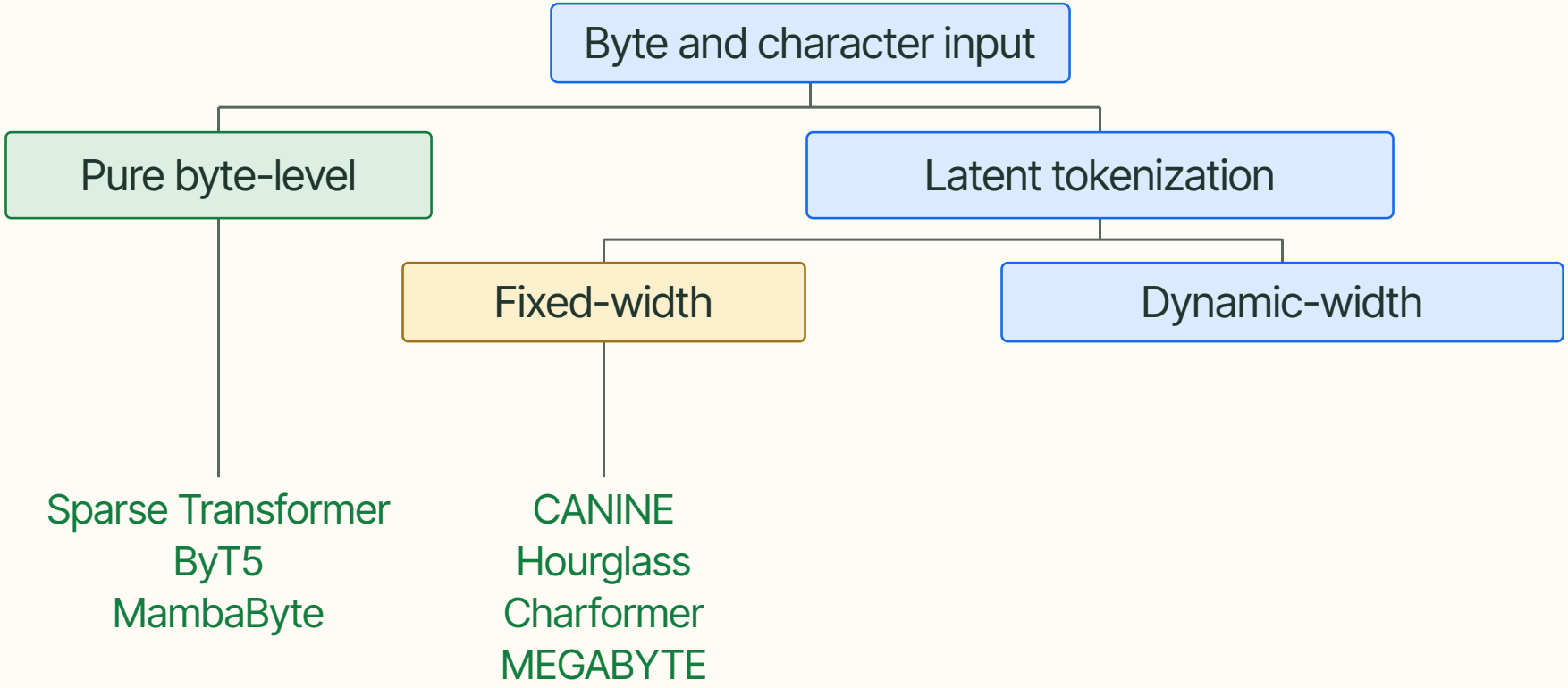
Byte and character model architectures



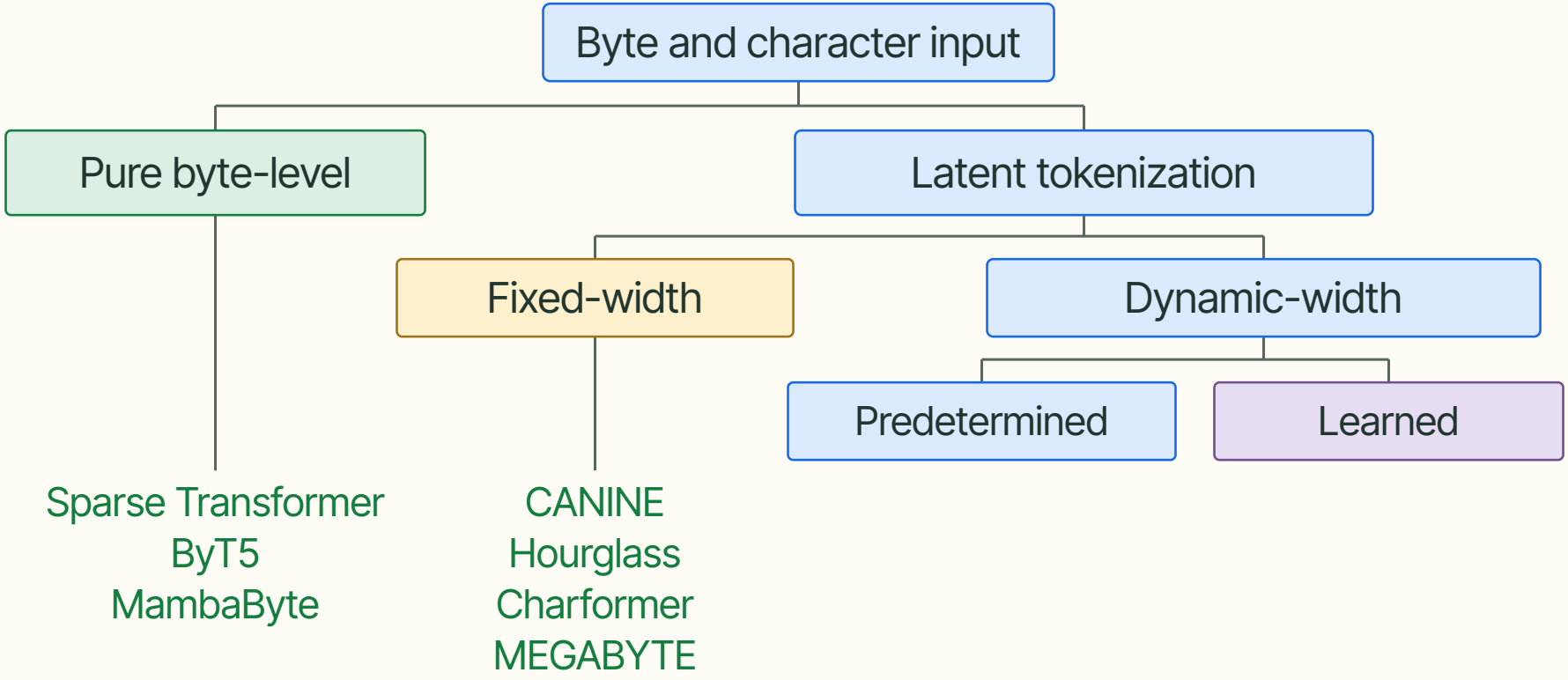
Byte and character model architectures



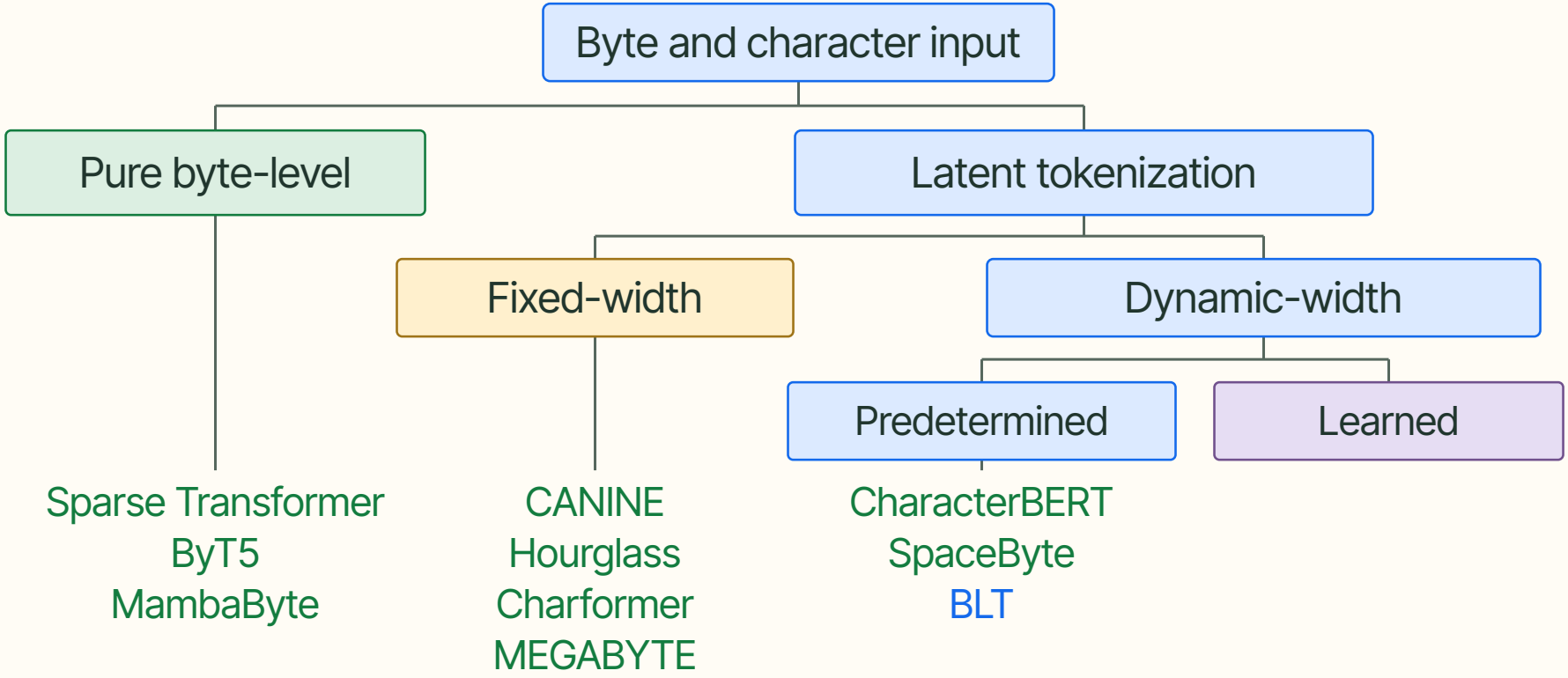
Byte and character model architectures



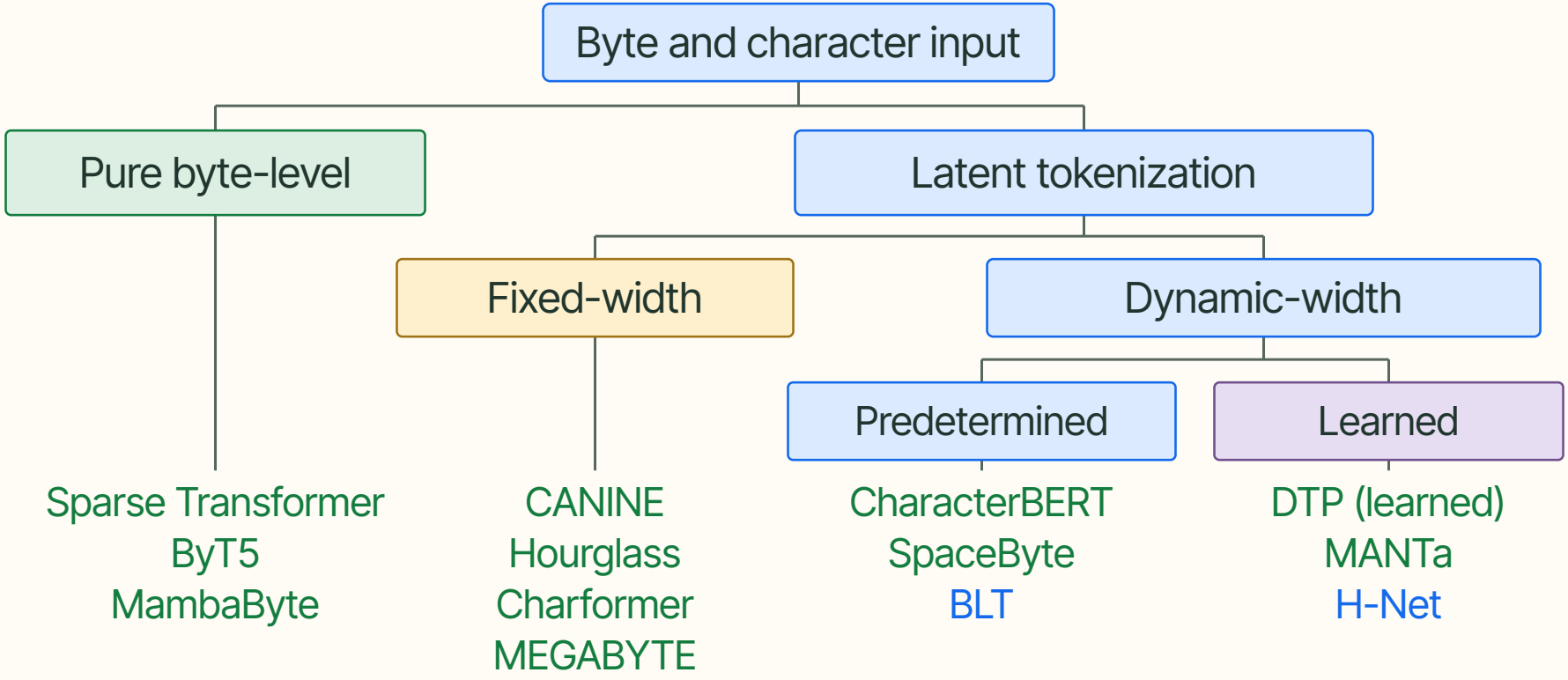
Byte and character model architectures



Byte and character model architectures

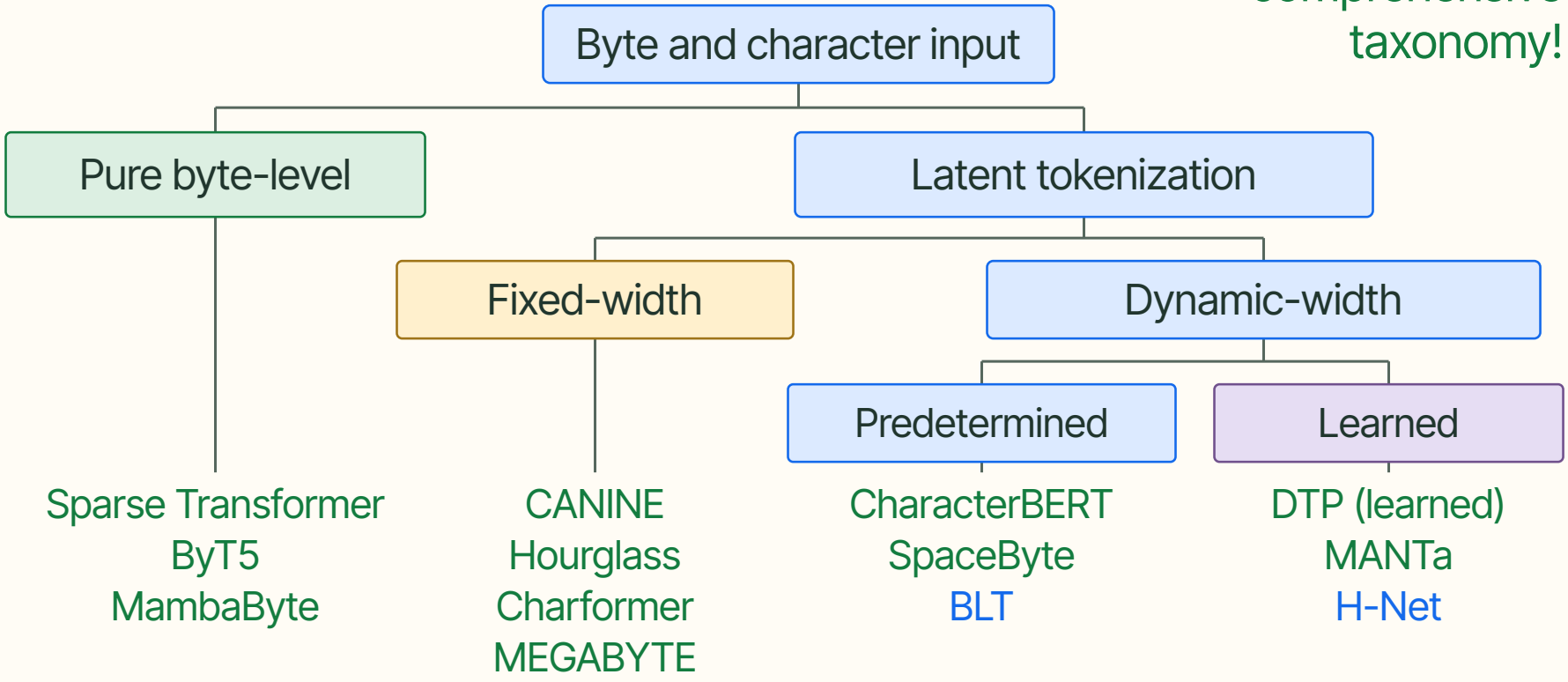


Byte and character model architectures



Byte and character model architectures

See survey section 9 for a comprehensive taxonomy!



Byte-level language modeling

A fixed alphabet of 256 byte values can represent UTF-8 text.

Text	Unicode code point	UTF-8 bytes (hex)
A	U+0041	41
é	U+00E9	C3 A9
€	U+20AC	E2 82 AC
😀	U+1F600	F0 9F 98 80

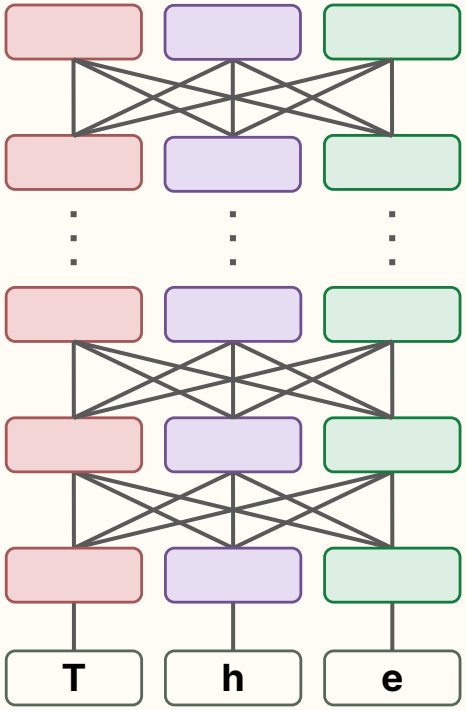
Byte-level language modeling

A fixed alphabet of 256 byte values can represent UTF-8 text.

Text	Unicode code point	UTF-8 bytes (hex)
A	U+0041	41
é	U+00E9	C3 A9
€	U+20AC	E2 82 AC
😀	U+1F600	F0 9F 98 80

Bytes and characters are different units.

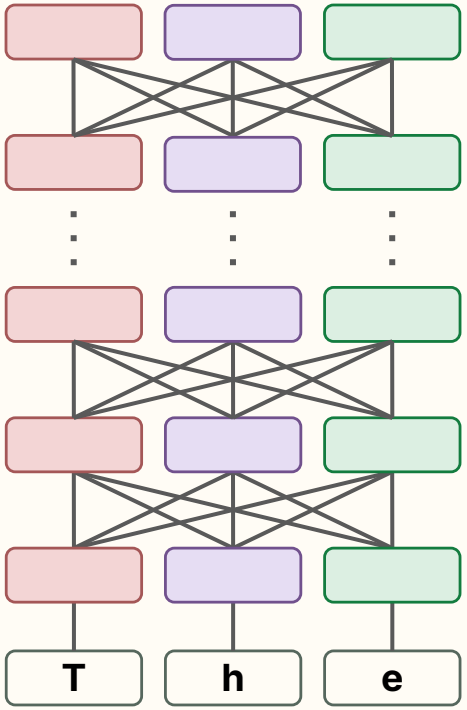
Pure byte language modeling with ByT5



- T5-style encoder–decoder with byte input and output.

Xue et al. (2022), Simplified encoder view.

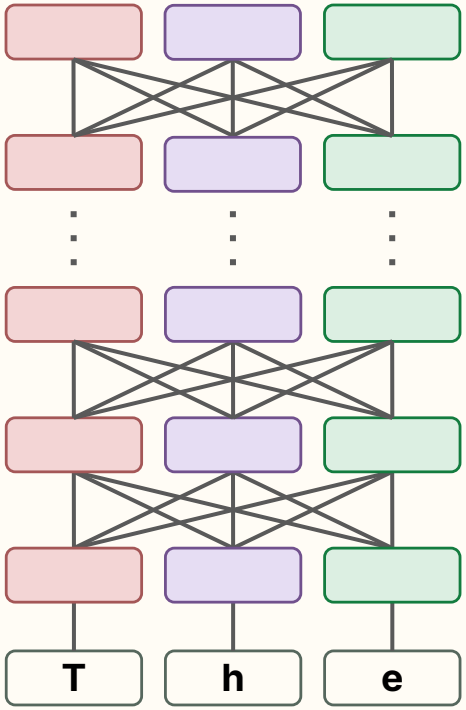
Pure byte language modeling with ByT5



- T5-style encoder–decoder with byte input and output.
- Pre-training reconstructs masked spans of bytes.

Xue et al. (2022), Simplified encoder view.

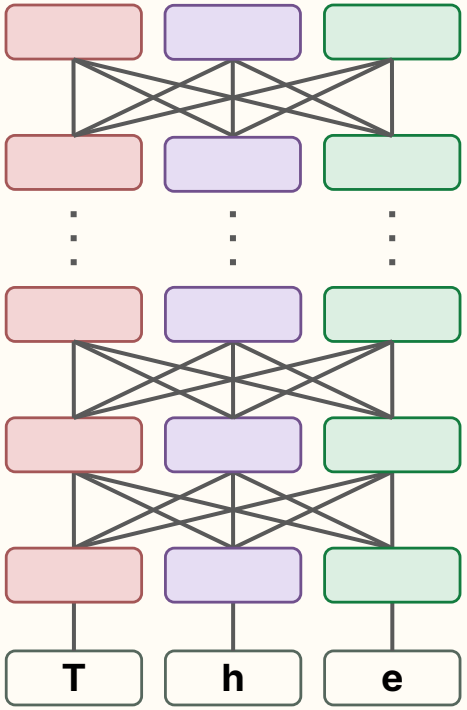
Pure byte language modeling with ByT5



- T5-style encoder–decoder with byte input and output.
- Pre-training reconstructs masked spans of bytes.
- The encoder keeps every input position throughout its layers.

Xue et al. (2022), Simplified encoder view.

Pure byte language modeling with ByT5



Xue et al. (2022), Simplified encoder view.

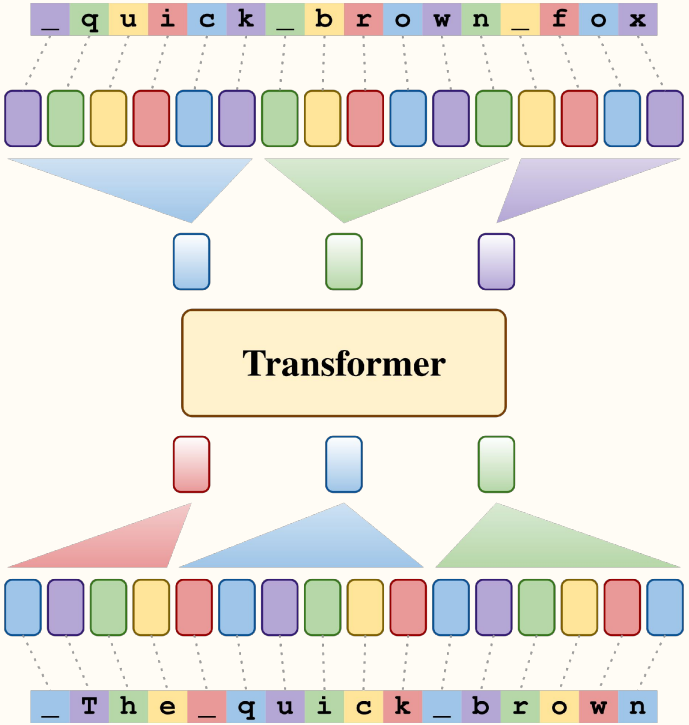
- T5-style encoder–decoder with byte input and output.
- Pre-training reconstructs masked spans of bytes.
- The encoder keeps every input position throughout its layers.

Do all later layers need every byte position?

Can we keep fine-grained access while concentrating expensive computation on fewer positions?

Latent Tokenization

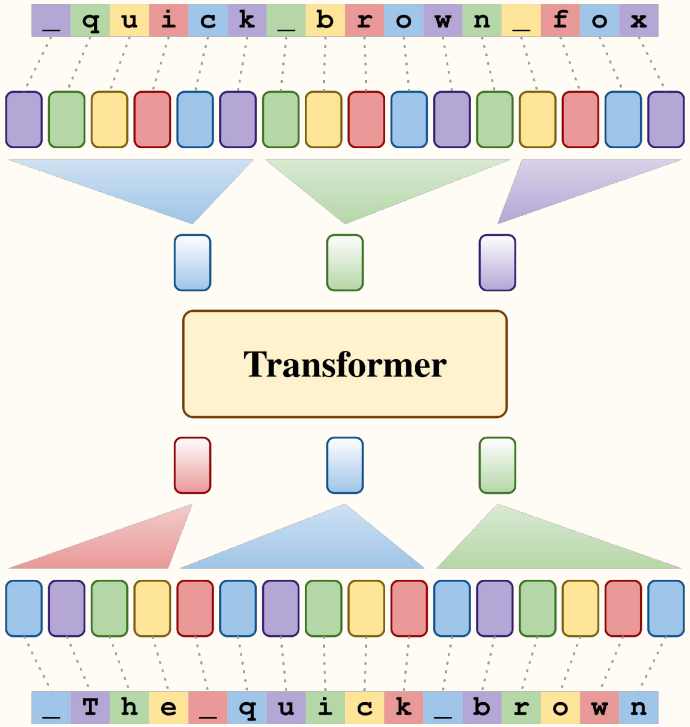
A general hierarchical model



- Fine-grained input: read every byte.

16 input bytes, 3 latent positions
Illustrative patch-level prediction

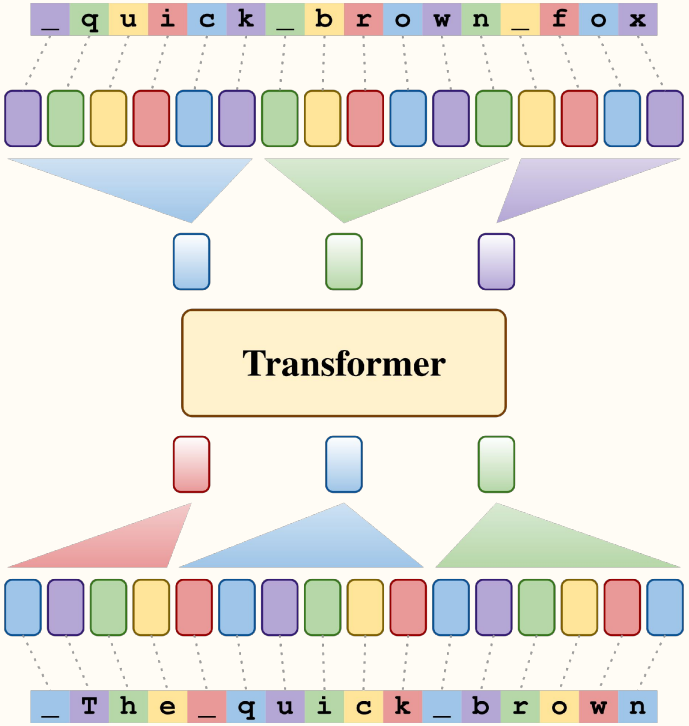
A general hierarchical model



- Fine-grained input: read every byte.
- Shorter latent sequence: concentrate the main computation on “latent tokens” or “patch representations”.

16 input bytes, 3 latent positions
Illustrative patch-level prediction

A general hierarchical model



- Fine-grained input: read every byte.
- Shorter latent sequence: concentrate the main computation on “latent tokens” or “patch representations”.
- Fine-grained output: reconstruct or condition byte predictions.

16 input bytes, 3 latent positions
Illustrative patch-level prediction

How to tokenize in latent space?

Rules and supervision

- Fixed-size segments: N bytes each (MEGABYTE).

How to tokenize in latent space?

Rules and supervision

- Fixed-size segments: N bytes each (MEGABYTE).
- Textual cues: use whitespace or spacelike bytes (SpaceByte, HAT).
- Entropy guidance: prediction uncertainty suggests boundaries (DTP, BLT).

How to tokenize in latent space?

Rules and supervision

- Fixed-size segments: N bytes each (MEGABYTE).
- Textual cues: use whitespace or spacelike bytes (SpaceByte, HAT).
- Entropy guidance: prediction uncertainty suggests boundaries (DTP, BLT).
- Target tokenizer: supervise boundaries to mimic a source tokenizer (Bolmo).

How to tokenize in latent space?

Rules and supervision

- Fixed-size segments: N bytes each (MEGABYTE).
- Textual cues: use whitespace or spacelike bytes (SpaceByte, HAT).
- Entropy guidance: prediction uncertainty suggests boundaries (DTP, BLT).
- Target tokenizer: supervise boundaries to mimic a source tokenizer (Bolmo).

End-to-end boundary learning

- A binary boundary predictor learns where to split the sequence (DTP, H-Net).
- Boundary decisions can be trained jointly with next-symbol prediction.

How to tokenize in latent space?

Rules and supervision

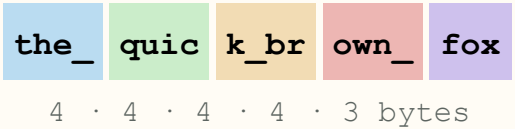
- Fixed-size segments: N bytes each (MEGABYTE).
- Textual cues: use whitespace or spacelike bytes (SpaceByte, HAT).
- Entropy guidance: prediction uncertainty suggests boundaries (DTP, BLT).
- Target tokenizer: supervise boundaries to mimic a source tokenizer (Bolmo).

End-to-end boundary learning

- A binary boundary predictor learns where to split the sequence (DTP, H-Net).
- Boundary decisions can be trained jointly with next-symbol prediction.
- Reinforcement learning can optimize segmentation decisions (Dauncey & Wattenhofer, 2026).

Static guidance: fixed segments or text cues

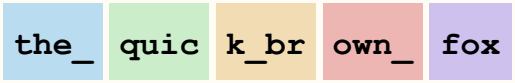
Fixed width



MEGABYTE
Uses a fixed patch size. Segments can span spaces, and boundaries can occur inside words.

Static guidance: fixed segments or text cues

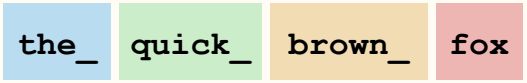
Fixed width



4 · 4 · 4 · 4 · 3 bytes

MEGABYTE
Uses a fixed patch size. Segments can span spaces, and boundaries can occur inside words.

Text-boundary cues

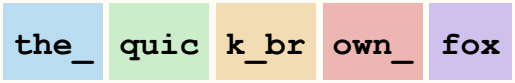


4 · 6 · 6 · 3 bytes

SpaceByte · HAT
Text cues define variable-length segments. The rule does not use prediction uncertainty.

Static guidance: fixed segments or text cues

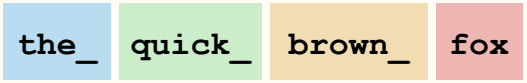
Fixed width



4 · 4 · 4 · 4 · 3 bytes

MEGABYTE
Uses a fixed patch size.
Segments can span spaces, and boundaries can occur inside words.

Text-boundary cues



4 · 6 · 6 · 3 bytes

SpaceByte · HAT
Text cues define variable-length segments.
The rule does not use prediction uncertainty.

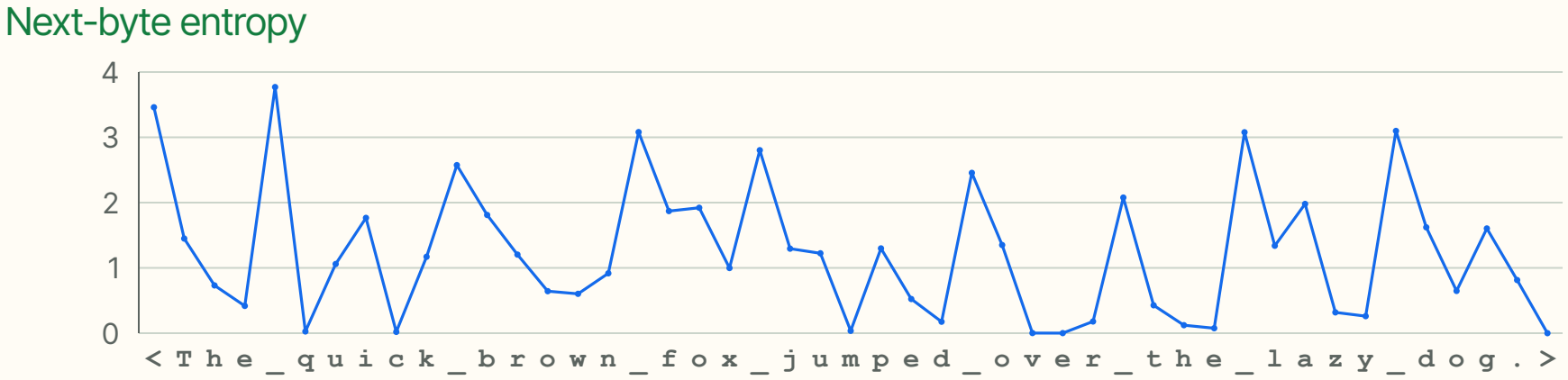
Trade-off

Cheap to compute, easy to train, and deterministic

But the boundary rule is fixed in advance, not adaptive...

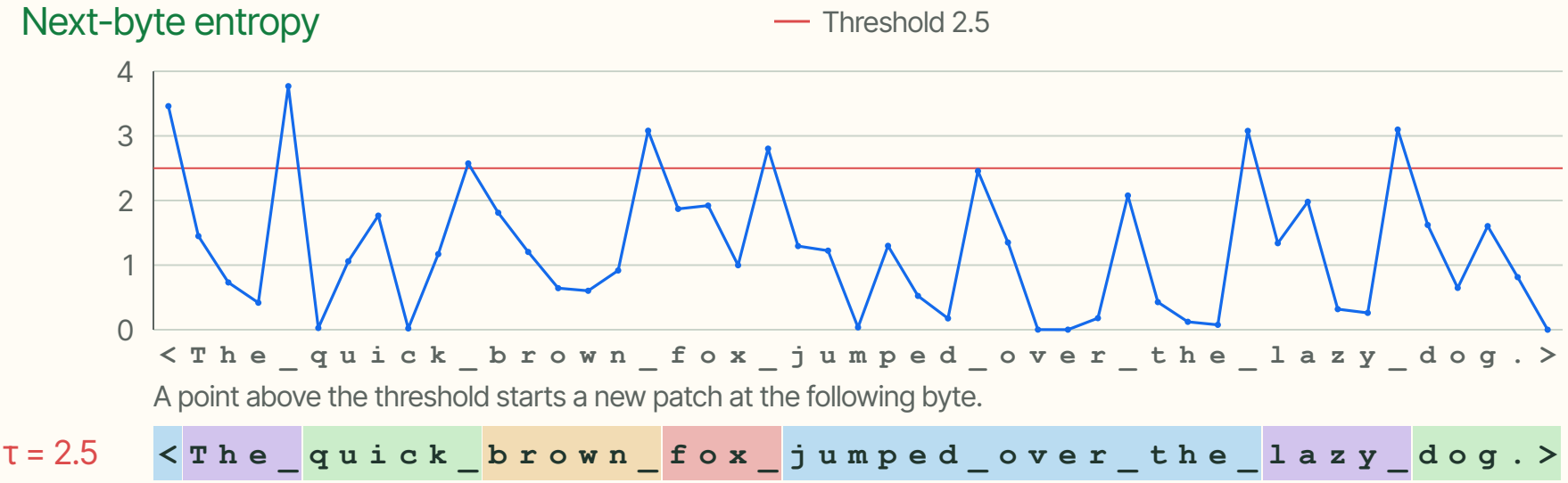
Next: entropy-guided boundaries

Entropy can suggest where to tokenize



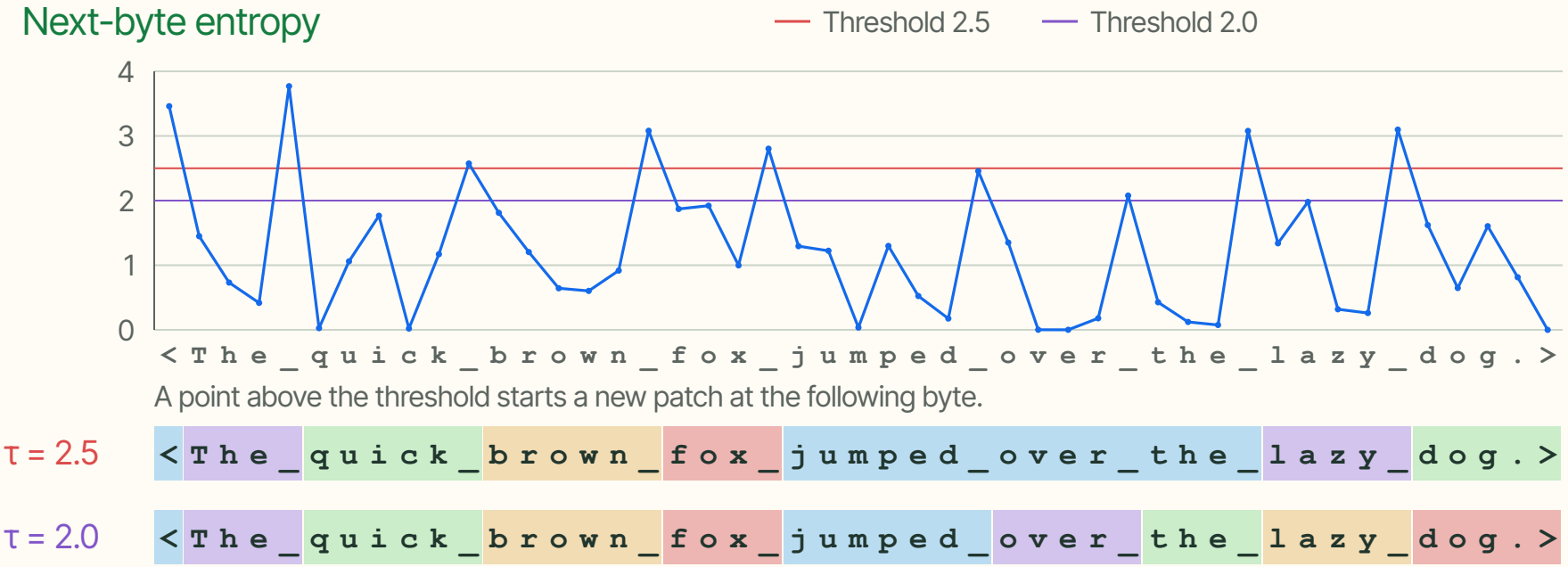
Cognetta et al. (2026), Example 9.2, p. 63. Entropies from BLT's entropy model.

Entropy can suggest where to tokenize



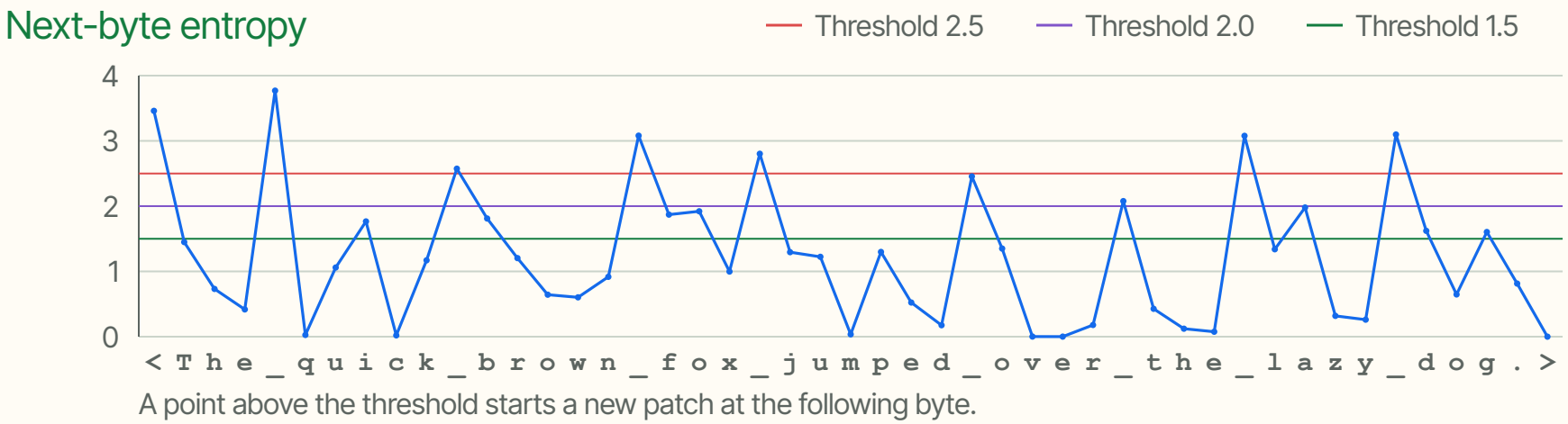
Cognetta et al. (2026), Example 9.2, p. 63. Entropies from BLT's entropy model.

Entropy can suggest where to tokenize



Cognetta et al. (2026), Example 9.2, p. 63. Entropies from BLT's entropy model.

Entropy can suggest where to tokenize



A point above the threshold starts a new patch at the following byte.

τ = 2.5 `< T h e _ q u i c k _ b r o w n _ f o x _ j u m p e d _ o v e r _ t h e _ l a z y _ d o g . >`

τ = 2.0 `< T h e _ q u i c k _ b r o w n _ f o x _ j u m p e d _ o v e r _ t h e _ l a z y _ d o g . >`

τ = 1.5 `< T h e _ q u i c k _ b r o w n _ f o x _ j u m p e d _ o v e r _ t h e _ l a z y _ d o g . >`

Cognetta et al. (2026), Example 9.2, p. 63. Entropies from BLT's entropy model.

Learned boundaries can imitate a tokenizer

One way to build a byte-level model is to train it to reproduce an existing tokenizer's boundaries.

1. Source tokenizer

Its boundaries define the training target.



Minixhofer et al. (2026), Methods. Illustrative target boundaries.

Learned boundaries can imitate a tokenizer

One way to build a byte-level model is to train it to reproduce an existing tokenizer's boundaries.

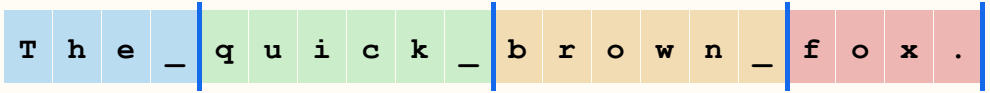
1. Source tokenizer

Its boundaries define the training target.



2. Byte model learns the segmentation

The boundary predictor marks token ends.



Minixhofer et al. (2026), Methods. Illustrative target boundaries.

Learned boundaries can imitate a tokenizer

One way to build a byte-level model is to train it to reproduce an existing tokenizer's boundaries.

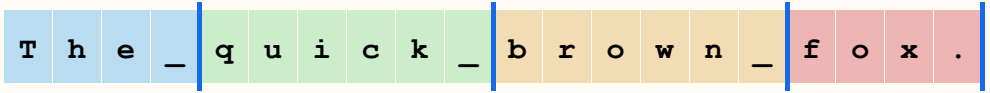
1. Source tokenizer

Its boundaries define the training target.



2. Byte model learns the segmentation

The boundary predictor marks token ends.



Why do this?

An existing tokenizer provides boundary targets. The byte-level model learns the segmentation while retaining access to individual bytes.

Minixhofer et al. (2026), Methods. Illustrative target boundaries.

Learned boundaries can imitate a tokenizer

One way to build a byte-level model is to train it to reproduce an existing tokenizer's boundaries.

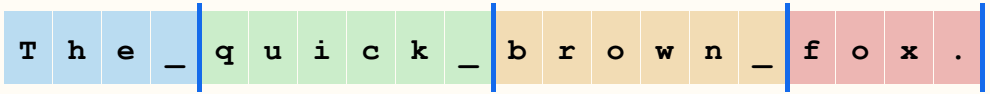
1. Source tokenizer

Its boundaries define the training target.



2. Byte model learns the segmentation

The boundary predictor marks token ends.



Why do this?

An existing tokenizer provides boundary targets. The byte-level model learns the segmentation while retaining access to individual bytes.

Contrast: BLT uses a separate entropy model; H-Net learns routing jointly.

Minixhofer et al. (2026), Methods. Illustrative target boundaries.

Learn the boundaries end-to-end

Boundary prediction

- A binary predictor scores each input position: boundary or no boundary.
- The global model processes one representation per patch.
- The local decoder returns to byte or character predictions.

Dauncey & Wattenhofer (2026); Hwang et al. (2026); Nawrot et al. (2023).

Learn the boundaries end-to-end

Boundary prediction

- A binary predictor scores each input position: boundary or no boundary.
- The global model processes one representation per patch.
- The local decoder returns to byte or character predictions.

Learning the policy

- DTP and H-Net learn chunking as part of hierarchical sequence modeling.
- Reinforcement-learning work treats the segmentation choice as an action.

Adaptive boundaries add an optimization challenge.

Dauncey & Wattenhofer (2026); Hwang et al. (2026); Nawrot et al. (2023).

Latent tokenization recap

Character access	Bytes remain available to local computation.	Character skills still have to be learned.

Latent tokenization recap

Character access	Bytes remain available to local computation.	Character skills still have to be learned.
Segmentation	Boundaries can be fixed, guided or learned.	A learned boundary is not guaranteed to be linguistically optimal.

Latent tokenization recap

Character access	Bytes remain available to local computation.	Character skills still have to be learned.
Segmentation	Boundaries can be fixed, guided or learned.	A learned boundary is not guaranteed to be linguistically optimal.
Efficiency	Global computation runs over fewer positions.	Local models and boundary prediction also cost compute.

Latent tokenization recap

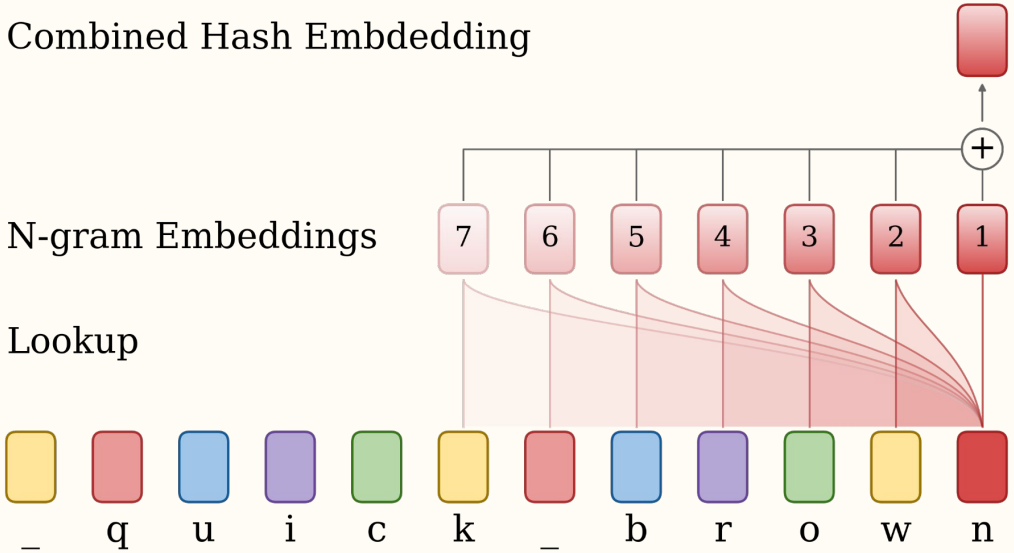
Character access	Bytes remain available to local computation.	Character skills still have to be learned.
Segmentation	Boundaries can be fixed, guided or learned.	A learned boundary is not guaranteed to be linguistically optimal.
Efficiency	Global computation runs over fewer positions.	Local models and boundary prediction also cost compute.
Vocabulary	No fixed subword vocabulary is required.	UTF-8 byte lengths still vary across scripts.

Efficiency Improvements

Hash Embeddings

Concern: Bytes limit ability to encode longer context

SOLUTION:
Use multiple lookup tables to encode spans of text without costly encoder layers



Long Sequence Processing

Concern: Computation and Memory cost of Attention is $O(T^2)$

True, but at larger scales feed forward dominates

FURTHER SOLUTIONS:

- **Linear attention (SSM or SWA)**
- **Flash attention (for Memory)**
- **Hierarchical architectures**

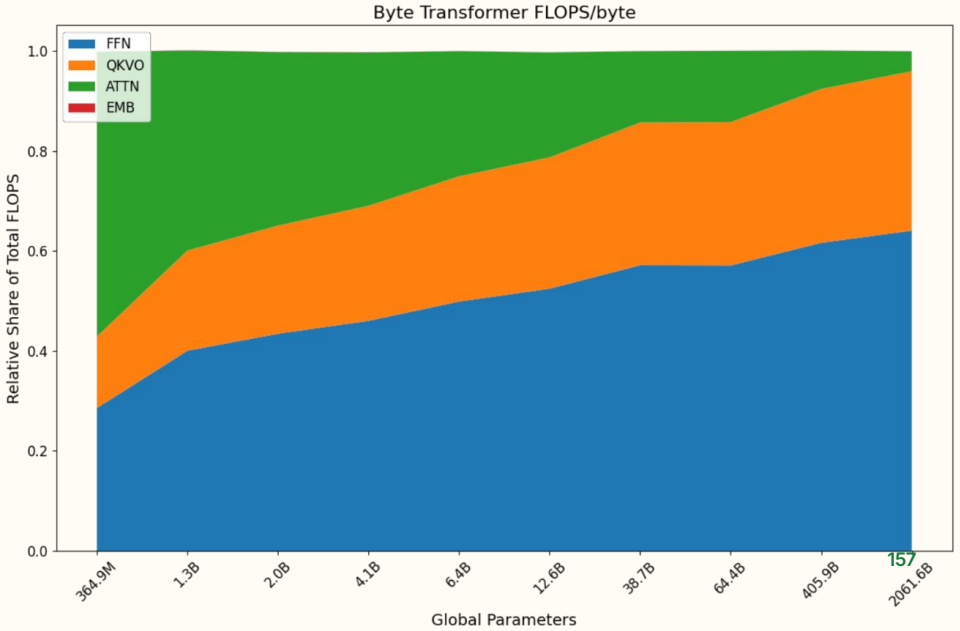
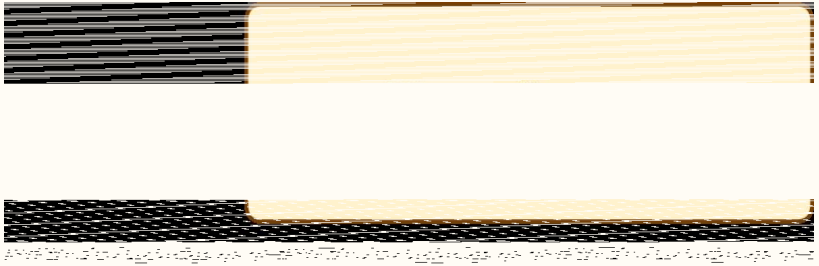


Figure Author: Artidoro Pagnoni

Efficient Decoding

Concern: Need to map from and into long byte sequence in decoding



SOLUTIONS:

Parallel decoding approaches:

- Speculative decoding
- Diffusion ⇢
- Multi token prediction

FastBLT, Kallini et al. 2026,
Multi Byte Prediction, Owodunni et al. 2026

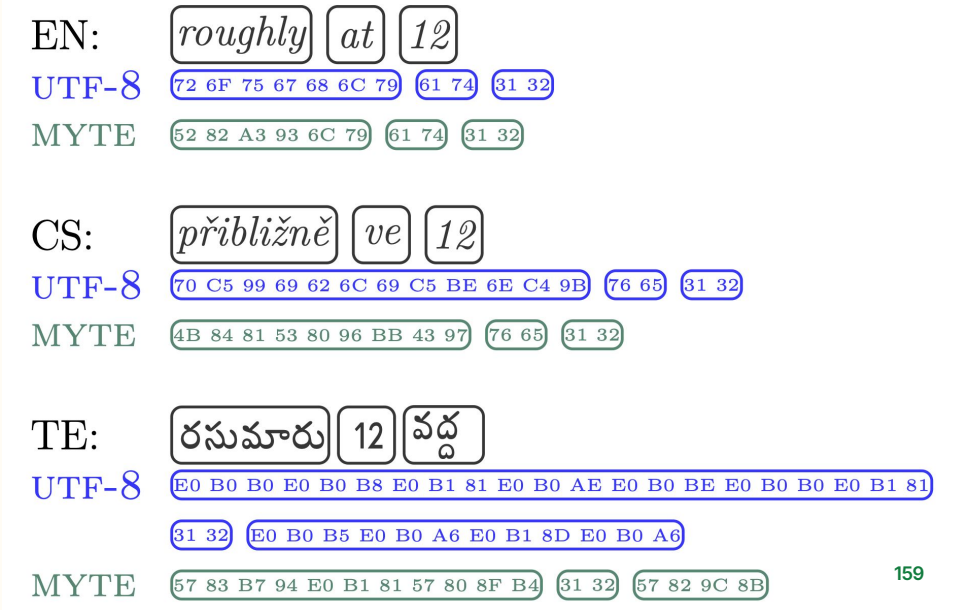
Bytes Are Not Fair

Concern: The same information in different languages is expressed by significantly unequal number of bytes (i.e., parity is far from 1)

SOLUTIONS:

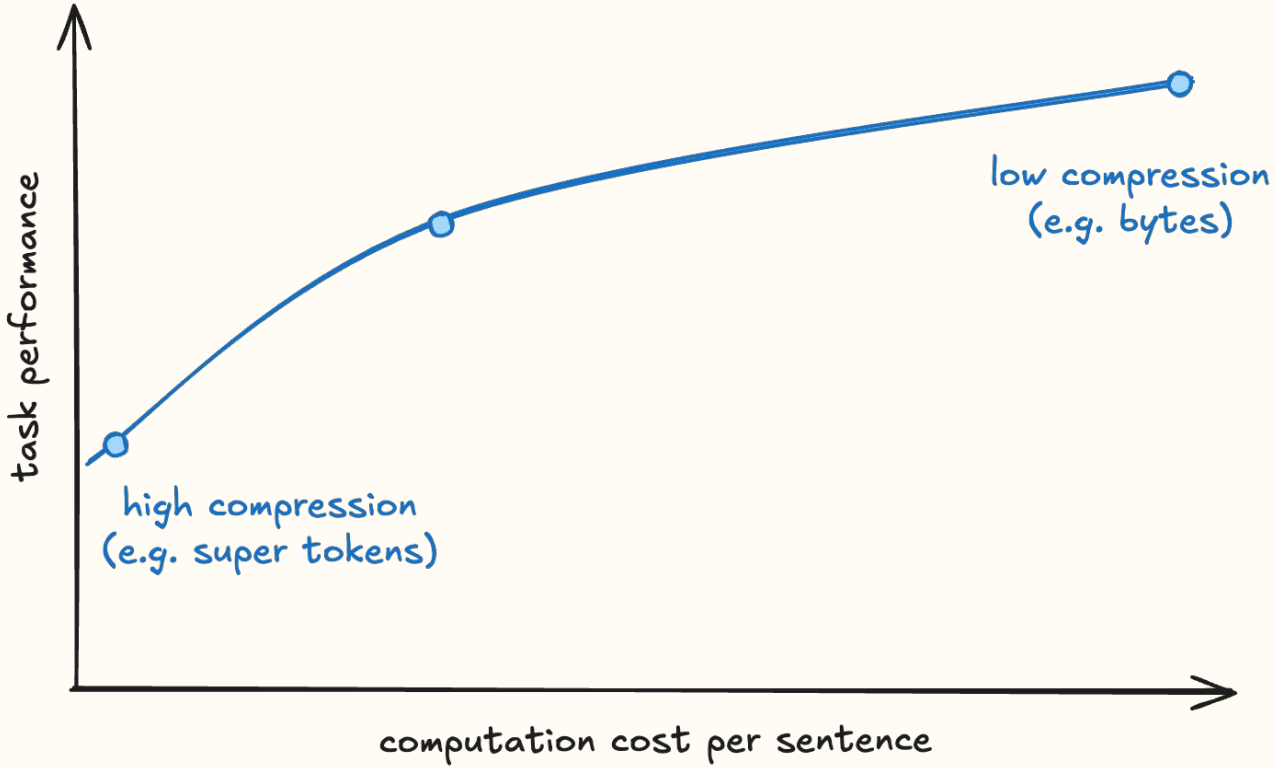
Changing encoding convention →

Language-adaptive compression in latent tokenization

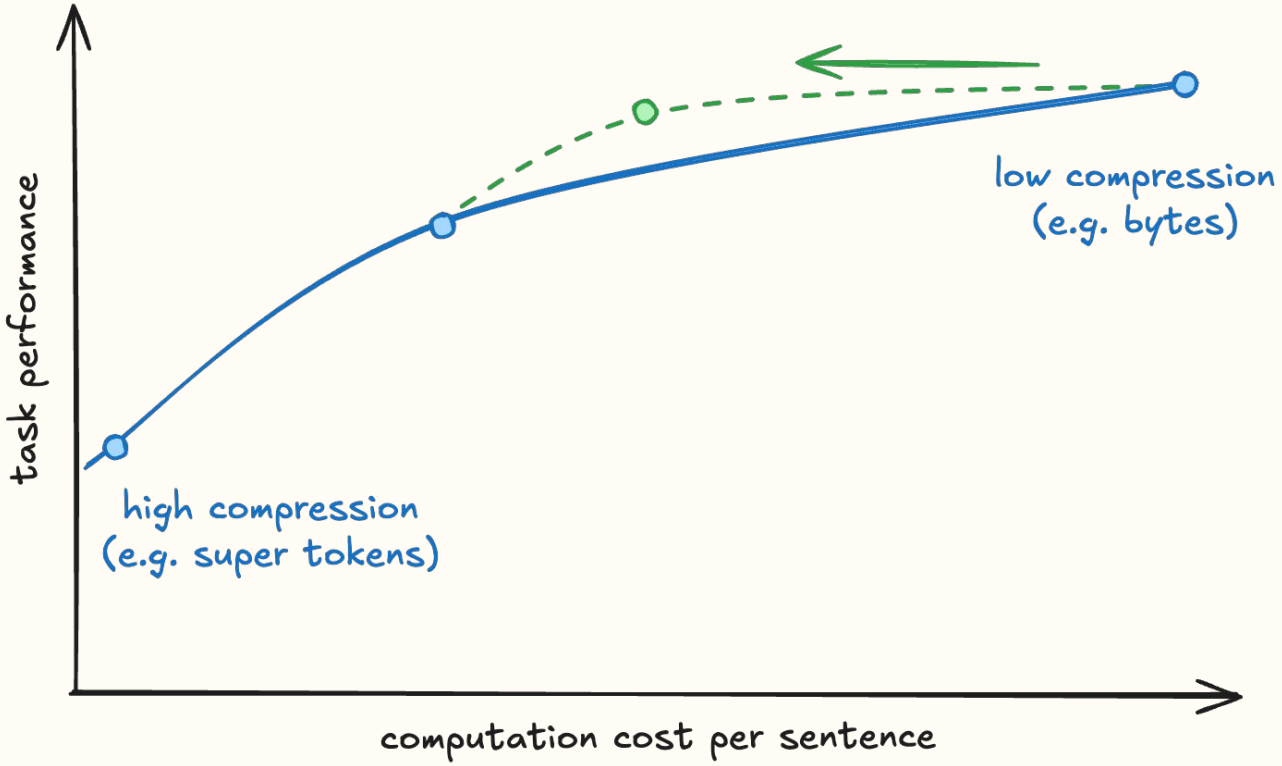


MYTE, Limisiewicz et al. 2024
MAGNET, Ahia et al. 2024

Performance vs. Efficiency Trade-off



Performance vs. Efficiency Trade-off



Promises and Challenges

PROMISES

- data efficient: best performance on fixed size of dataset
- flexibility allows adaptive compute allocation
- robustness to noise

CHALLENGES

- inference speed
- long context performance
- not attested in post-training and frontier models (yet)

Are Bytes Fair?

Length of byte representation largely varies across languages, e.g.:

Malayalam: **3.1**

Georgian: **2.9**

x English

Hindi: **2.6**

These inequalities are inherited by tokenization method building on top of bytes, e.g. **BPE**

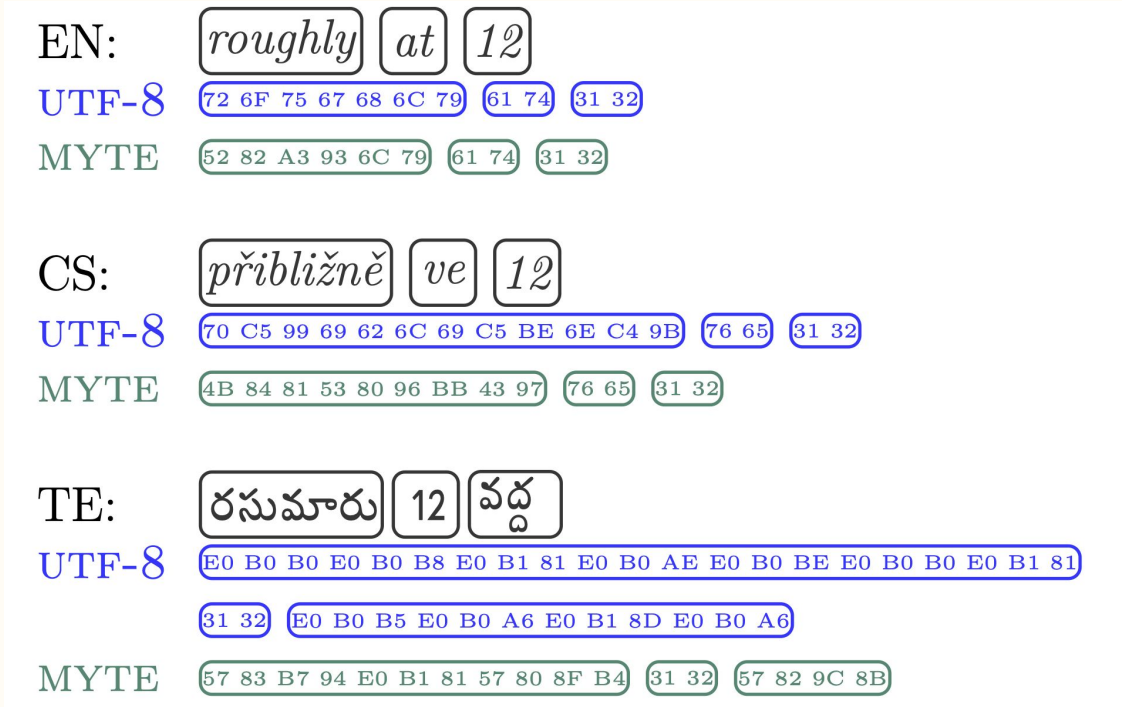
Are Bytes Fair?

MOTIVATIONS:

Fairness: Similar Sequence Lengths for the Same Information Across Languages

Efficiency: Shorter Sequence Lengths

💡 **Idea:** Morphemes* are basic unit of information, their number should be comparable across languages



Limisiewicz et al. 2024

Text as vision:
Tokenization with rendered text

Units of tokenization

Note: the term 'character' is very overloaded. Be careful to check what it refers to in a given paper!

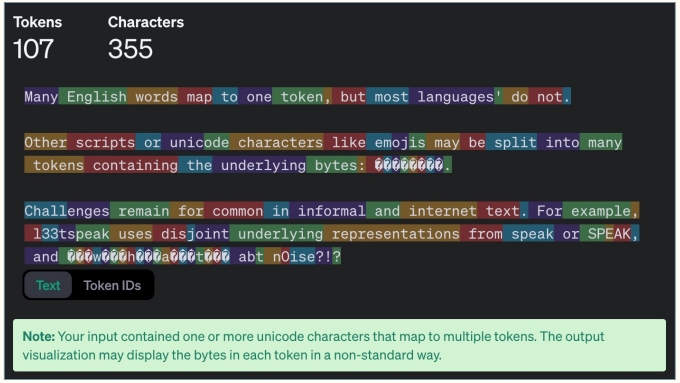
Café :

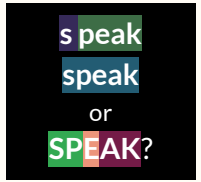
UTF-8 Bytes	43	61	66	65	CC 81
Codepoints	U+0043	U+0061	U+0066	U+0065	U+0301
Characters	C	a	f	e	◌́
Graphemes	C	a	f	é	

Recall: Tokenization woes

Note: some phenomena can be addressed by pretokenization or text normalization, but not all. Mistakes here are costly!!

- Textual noise
- Capitalization, punctuation
- Inserted characters such as diacritics
- …pretokenization is hard to get right! What if we didn't need it?





WIPO

The invention belongs to the field of biotechnology, pharmaceutics and medicine, it could be applied for the production of drugs and for the realization of medicinal technologies, particularly for the immunotherapy of oncological diseases.

Cyrillic Latin

Phenomena	Word	BPE	
Vowelization	كتاب الكِتابُ	كتاب ا ٠ ل ٠ ك ٠ ِ ٠ ت ٠ ا ٠ ب ٠ ُ	(1) (5)
Misspelling	language langauge	language la · ng · au · ge	(1) (4)
Visually Similar Characters	really rea11y	really re · a · 1 · 1 · y	(1) (5)
Shared Character Components	확인한다 확인했다	확인 · 한 · 다 확인 · 했다	(3) (2)

Examples of common behavior which cause divergent representations for subword models

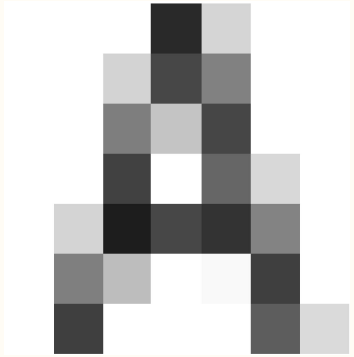
Robustness

a 0061 LATIN SMALL LETTER A	ɑ 0251 LATIN SMALL LETTER ALPHA	α 03B1 GREEK SMALL LETTER ALPHA	а 0430 CYRILLIC SMALL LETTER A	𝛂 237A APL FUNCTIONAL SYMBOL ALPHA	𝐚 1D41A MATHEMATICAL BOLD SMALL A
𝑎 1D44E MATHEMATICAL ITALIC SMALL A	𝒂 1D482 MATHEMATICAL BOLD ITALIC SMALL A	𝒶 1D4B6 MATHEMATICAL SCRIPT SMALL A	𝓪 1D4EA MATHEMATICAL BOLD SCRIPT SMALL A	𝔞 1D51E MATHEMATICAL FRAKTUR SMALL A	𝕒 1D552 MATHEMATICAL DOUBLE-STRUCK SMALL A
𝖆 1D586 MATHEMATICAL BOLD FRAKTUR SMALL A	𝖺 1D5BA MATHEMATICAL SANS-SERIF SMALL A	𝗮 1D5EE MATHEMATICAL SANS-SERIF BOLD SMALL A	𝘢 1D622 MATHEMATICAL SANS-SERIF ITALIC SMALL A	𝙖 1D656 MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL A	𝚊 1D68A MATHEMATICAL MONOSPACE SMALL A
𝛂 1D6C2 MATHEMATICAL BOLD SMALL ALPHA	𝛼 1D6FC MATHEMATICAL ITALIC SMALL ALPHA	𝜶 1D736 MATHEMATICAL BOLD ITALIC SMALL ALPHA	𝝰 1D770 MATHEMATICAL SANS-SERIF BOLD SMALL ALPHA	𝞪 1D7AA MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL ALPHA	ａ FF41 FULLWIDTH LATIN SMALL LETTER A

Rendering text

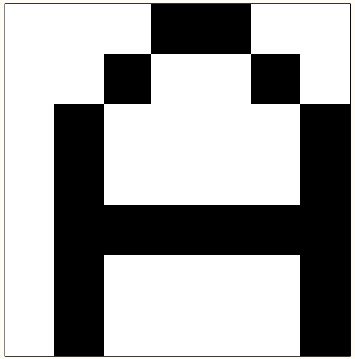


Letter A,
font size 5



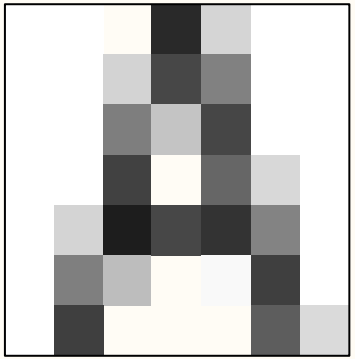
Rendering text

Bitmap font



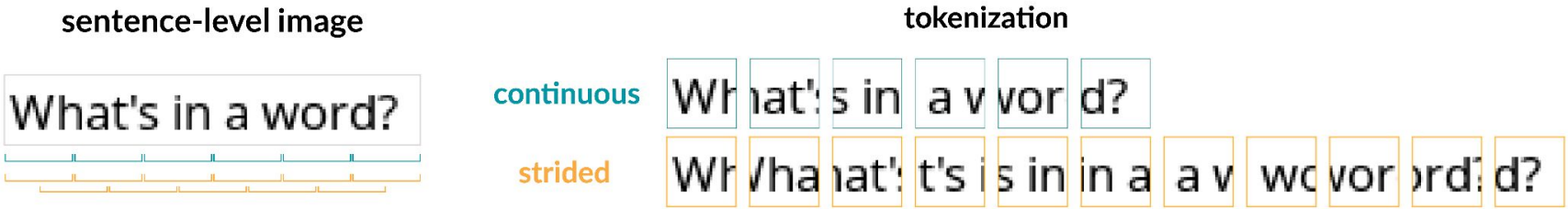
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	1	1	1	1	1	1
0	1	0	0	0	0	1
0	1	0	0	0	0	1
0	1	0	0	0	0	1

Vector font



255	255	255	41	213	255	255
255	255	211	71	129	255	255
255	255	126	196	70	255	255
255	255	65	253	102	216	255
255	212	29	71	51	131	255
255	127	189	255	249	63	255
255	63	255	255	255	93	218

Rendering text

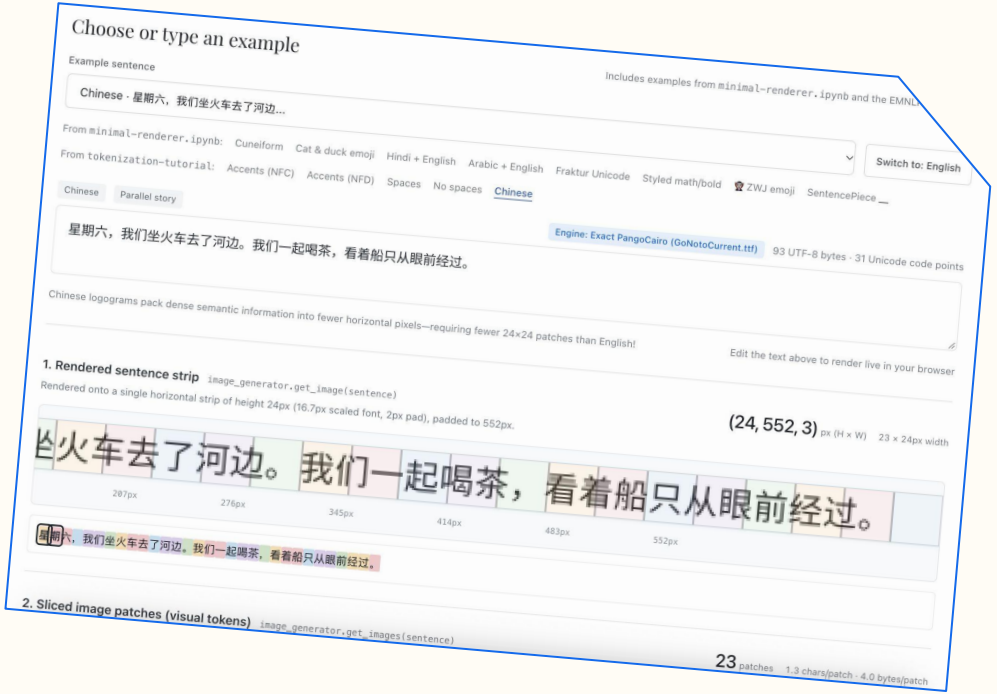


Rendering at the sentence-level and then tokenizing (or “patching”) means maintaining character shapes and spatial relationships

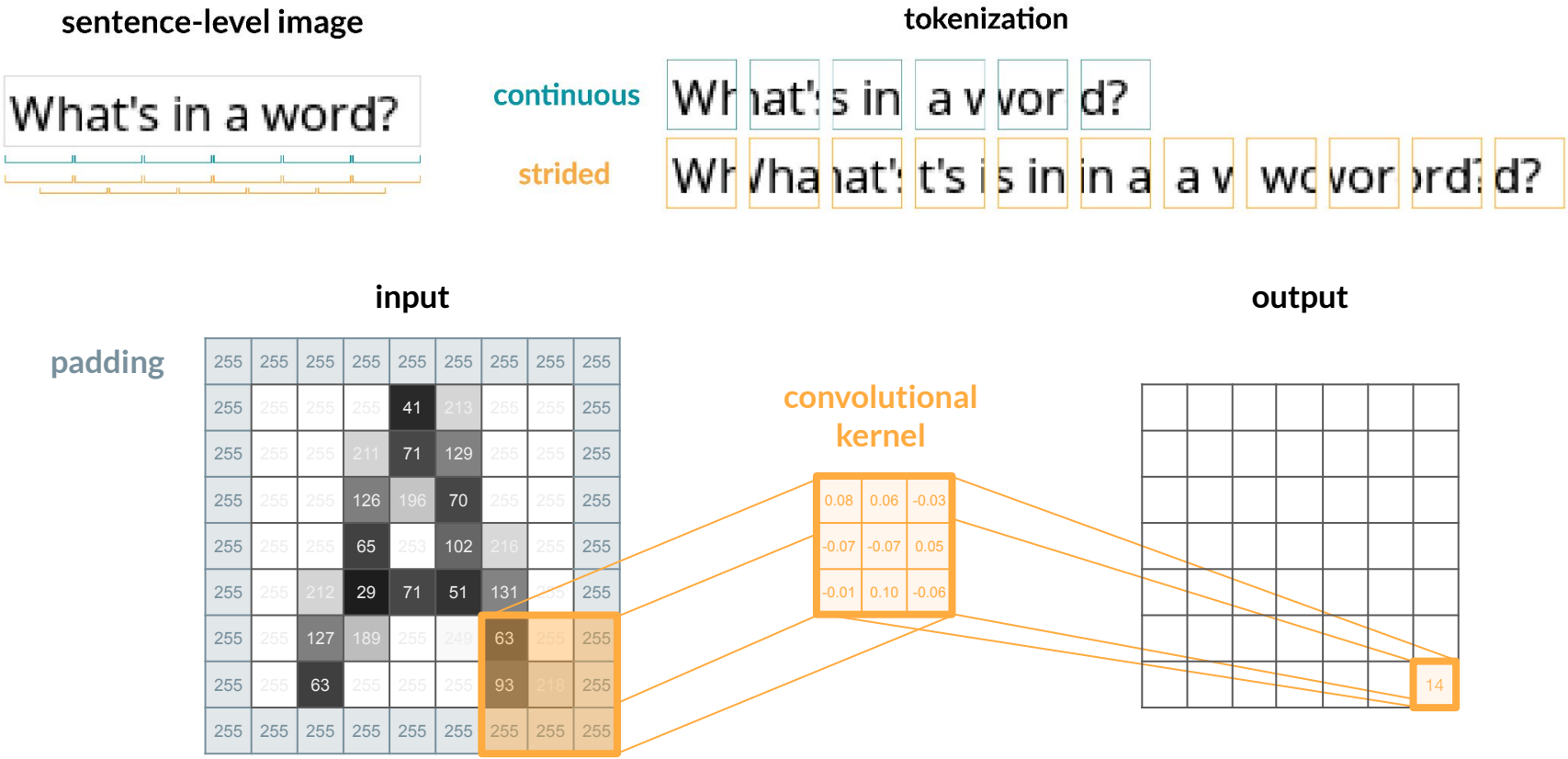
Salesky et al. (2021) Rust et al. (2023) Tschannen et al. (2023) Salesky et al. (2023) Tai et al. (2024) Chai et al. (2024) Lotz et al. (2025)

Interactive Demo

https://tokenization-tutorial.github.io/renderer/

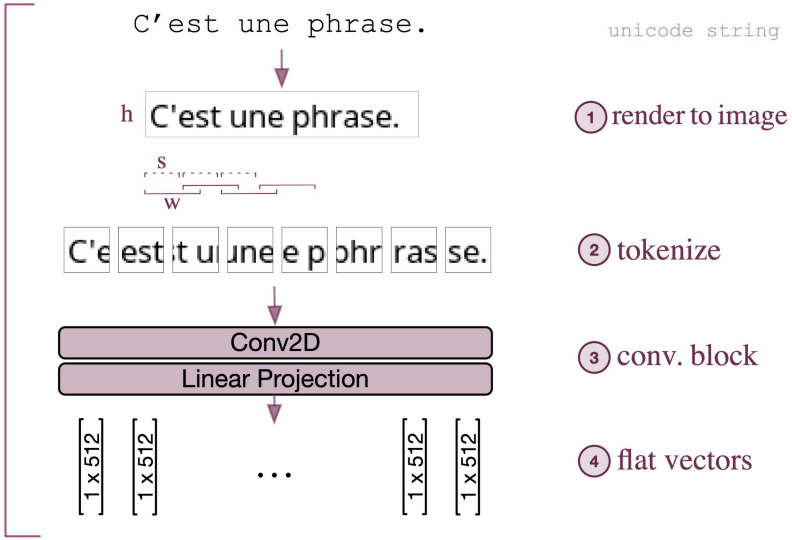


Tokenizing rendered text



Language modeling with visual representations

Visual embedder

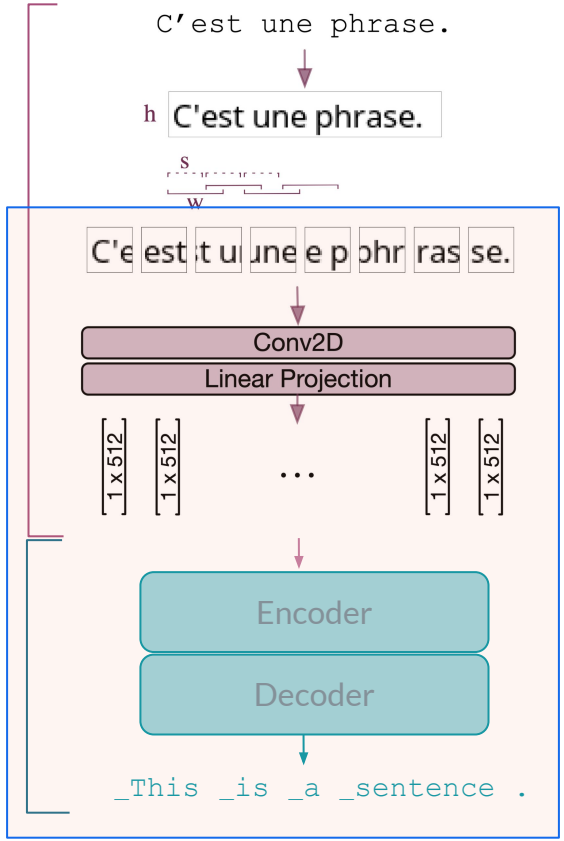


Parameters:
~200k,
constant wrt V

<1% of total model parameters

Language modeling with visual representations

Visual embedder



Parameters:
~200k,
constant wrt V

<1% of total model parameters

Standard Transformer

The rest of the model can proceed as usual *

Benefits:

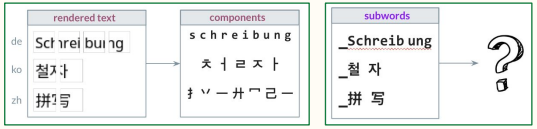
Robustness

Visually similar characters, but also insertions, substitutions as the model has access to token composition

ar-en

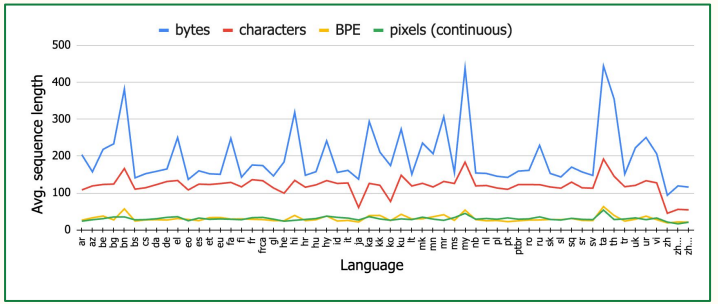
src	أنا كندية، وأنا أصغر أخواتي السبعة
noised	أنا كَنَدِيّة ، وَأَنا أَصْغَرِ إِخْوانِي السَبْعَة
ref	I'm Canadian, and I'm the youngest of seven kids.

Salesky et al (2021)



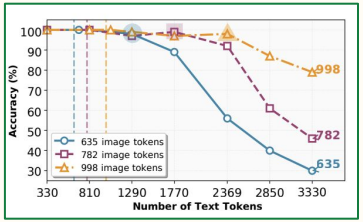
Sequence lengths

Equitable across languages and scripts



Data-efficiency / text compression

Both in transfer learning and for long-context text compression



Reduce tokens by ~50% & maintain performance
Li et al (2025)

Transfer learning and finetuning

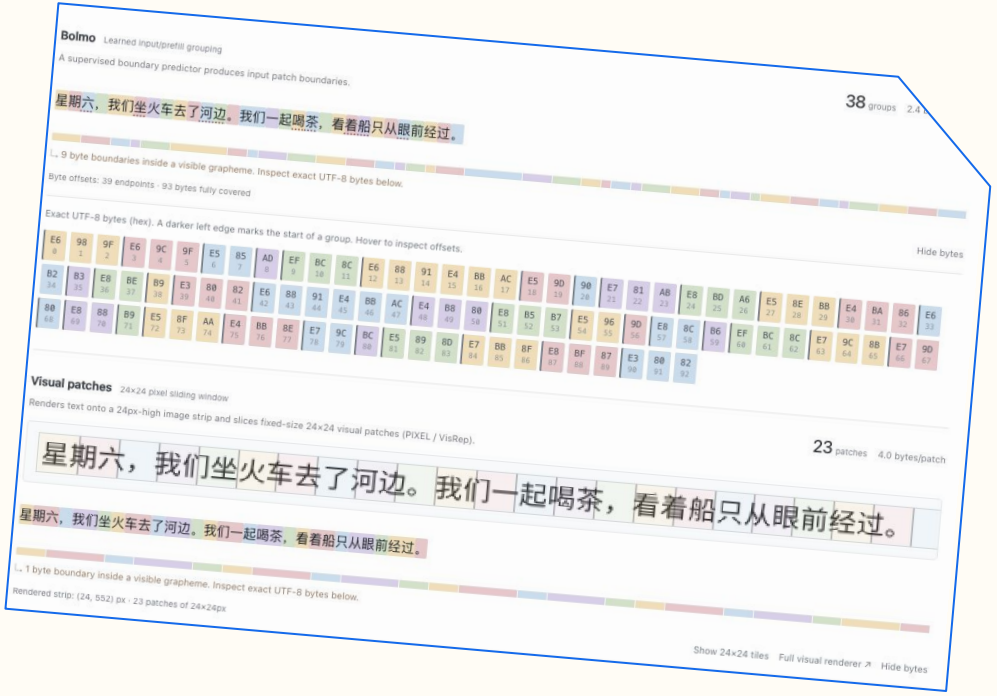
Directly finetune on unseen scripts, no transliteration or modification required

Language	ISO	Script	Unicode Range	Sample
English	eng	Latin		Universal Declaration of Human Rights
Assamese	asm	Bengali–Assamese		মানৱ অধিকাৰৰ সাৰ্বজনীন ঘোষণা
Bengali	ben	Bengali–Assamese		মানবাধিকারের সর্বজনীন ঘোষণা
Gujarati	guj	Gujarati		માનવ અધિકારોની સાર્વત્રિક ઘોષણા
Hindi	hin	Devanagari		मानव अधिकारों का सार्वजनिक घोषणापत्र
Kannada	kan	Kannada		ಮಾನವ ಹಕ್ಕುಗಳ ಸಾರ್ವತ್ರಿಕ ಘೋಷಣೆ
Malayalam	mal	Malayalam		മനുഷ്യാവകാശങ്ങളുടെ സാർവത്രിക പ്രഖ്യാപനം
Marathi	mar	Devanagari		मानवी हक्कांची सार्वत्रिक घोषणा
Odia (Oriya)	ory	Odia		ମାନବିକ ଅଧିକାରର ସର୍ବଜନୀନ ଘୋଷଣା	
Punjabi	pan	Gurmukhī		ਮਨੁੱਖੀ ਅਧਿਕਾਰਾਂ ਦਾ ਵਿਸ਼ਵਵਿਆਪੀ ਐਲਾਨਨਾਮਾ
Tamil	tam	Tamil		மனித உரிமைகளின் உலகளாவிய பிரகடனம்
Telugu	tel	Telugu		మానవ హక్కుల సార్వత్రిక ప్రకటన

Rust et al (2023)
Salesky et al (2023)
Kesen et al (2025)

Compare to other methods in the playground

https://tokenization-tutorial.github.io/playground



Questions?


That's all Folks!