Parity-Aware Byte-Pair Encoding: Improving Cross-lingual Fairness in Tokenization

Negar Foroutan[1*] **Clara Meister**[2*] **Debjit Paul**[1]
Joel Niklaus[3] **Sina Ahmadi**[4] **Antoine Bosselut**[1†] **Rico Sennrich**[4†]

[1]EPFL [2]ETH Zurich [3]Niklaus.ai [4]University of Zurich

Presented by Tarek Naous

An Overview of BPE for Tokenization

BPE Phases for Text Tokenization

Training:

- Pre-tokenization
  - Assumptions and heuristics before encoding (split on white spaces)

BPE Phases for Text Tokenization

Training:

- Pre-tokenization
  - Assumptions and heuristics before encoding (split on white spaces)

- Vocabulary Initialization
  - Initialize with all unique characters after pre-tokenization

BPE Phases for Text Tokenization

Training:

- Pre-tokenization
  - Assumptions and heuristics before encoding (split on white spaces)
- Vocabulary Initialization
  - Initialize with all unique characters after pre-tokenization
- Learning Merge Rules
  - Form sub-words to maximize compression (frequency-based objective)

BPE Phases for Text Tokenization

Training:

- Pre-tokenization
  - Assumptions and heuristics before encoding (split on white spaces)
- Vocabulary Initialization
  - Initialize with all unique characters after pre-tokenization
- Learning Merge Rules
  - Form sub-words to maximize compression (frequency-based objective)

Inference:

- Inference Algorithm
  - Decide how to tokenize by applying the merge rules

Training Corpus

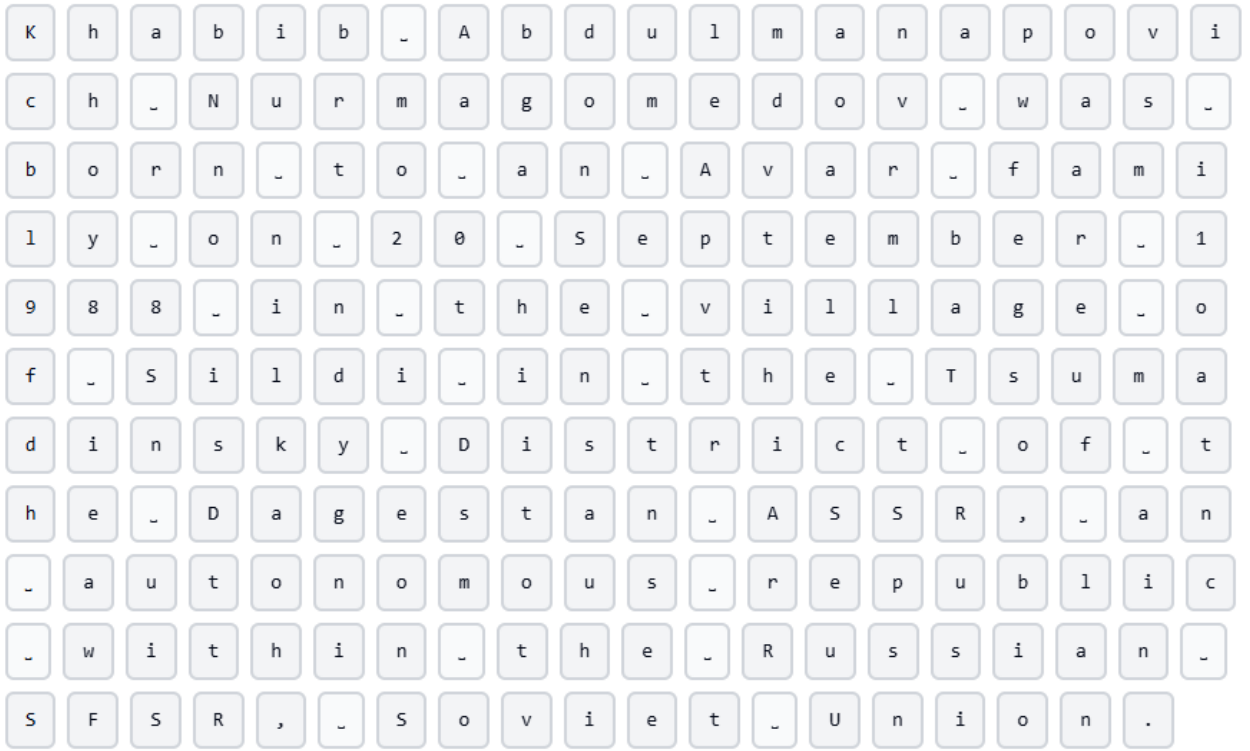
Khabib Abdulmanapovich Nurmagomedov was born to an Avar family on 20 September 1988 in the village of Sildi in the Tsumadinsky District of the Dagestan ASSR, an autonomous republic within the Russian SFSR, Soviet Union.

Training Corpus

Khabib Abdulmanapovich Nurmagomedov was born to an Avar family on 20 September 1988 in the village of Sildi in the Tsumadinsky District of the Dagestan ASSR, an autonomous republic within the Russian SFSR, Soviet Union.

Pre-tokenization

Split words on whitespace (_) to determine word boundaries (i.e., tokens cannot cross word boundaries; we either get sub-words or one word as a token)

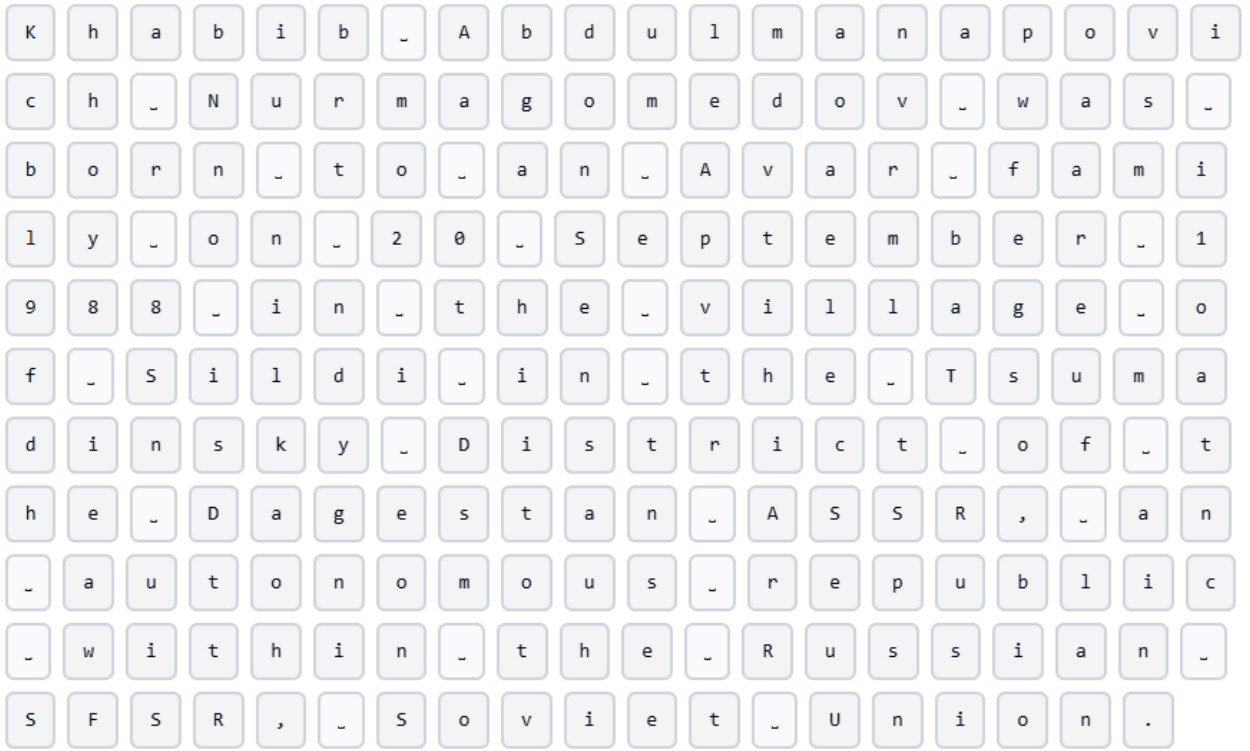


Training Corpus

Khabib Abdulmanapovich Nurmagomedov was born to an Avar family on 20 September 1988 in the village of Sildi in the Tsumadinsky District of the Dagestan ASSR, an autonomous republic within the Russian SFSR, Soviet Union.

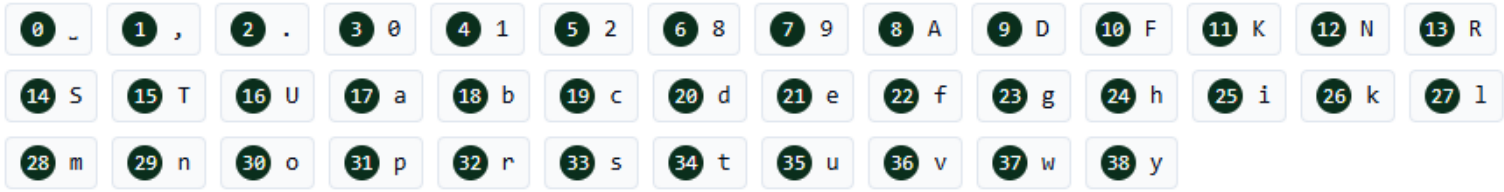
Pre-tokenization

Split words on whitespace (_) to determine word boundaries (i.e., tokens cannot cross word boundaries; we either get sub-words or one word as a token)



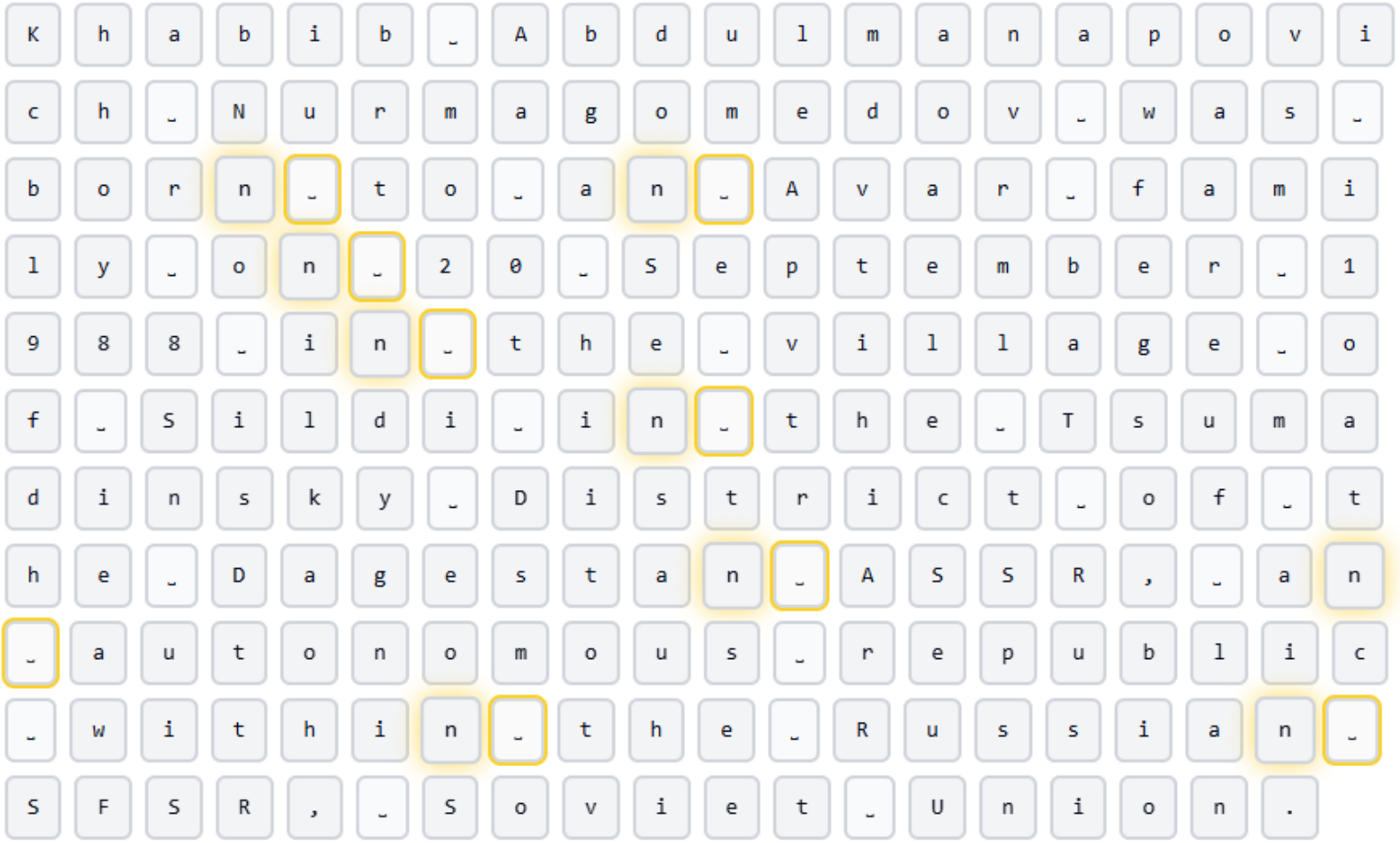
Vocab Initialization

Place all unique characters in the vocabulary

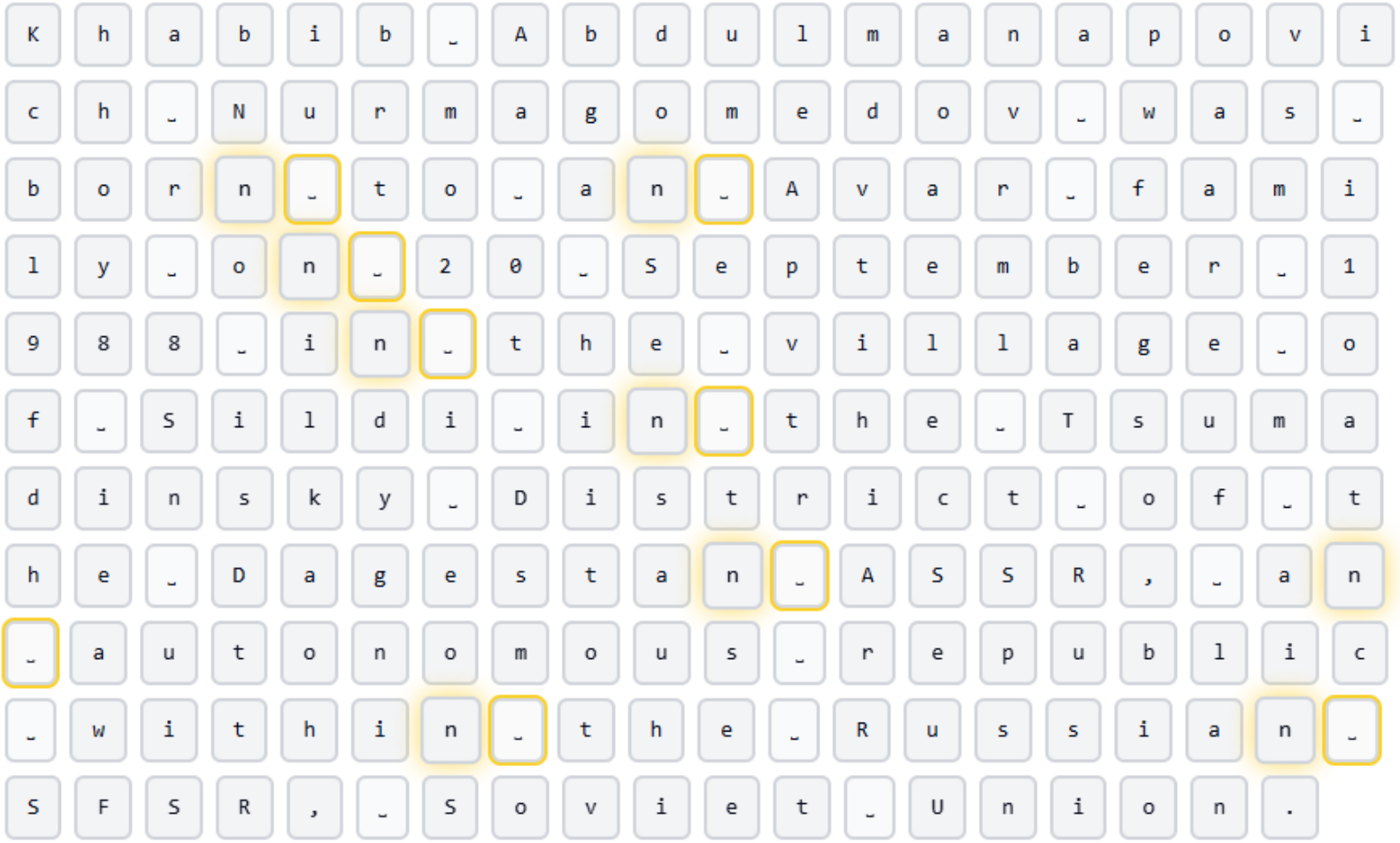


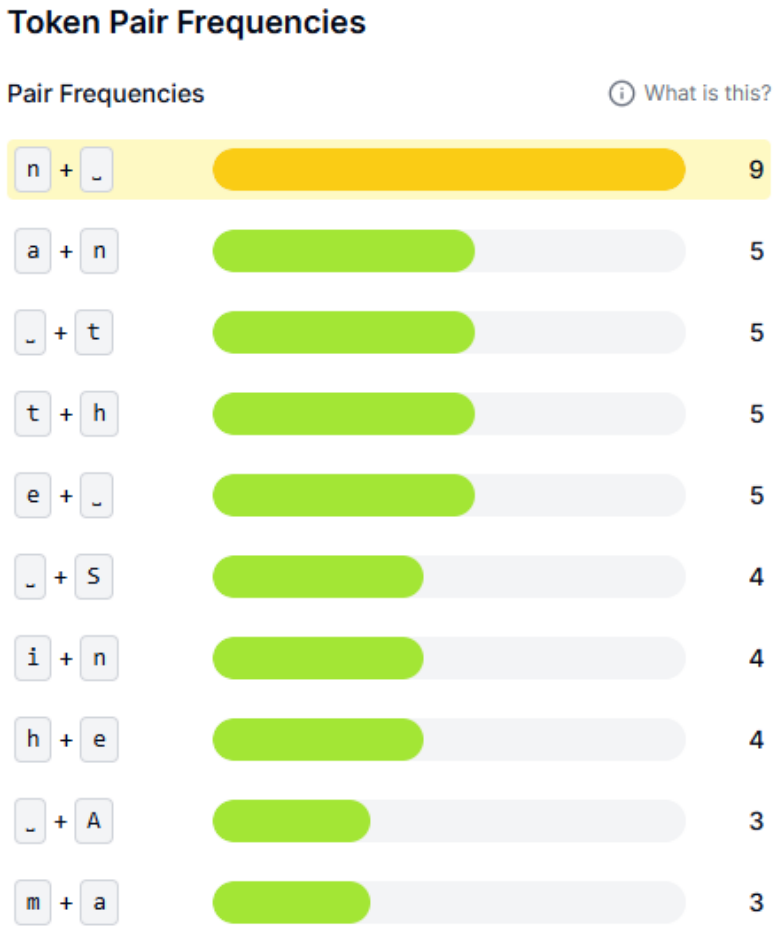
Merge Rules: Step 1 – Count pair frequencies

Merge Rules: Step 1 – Count pair frequencies



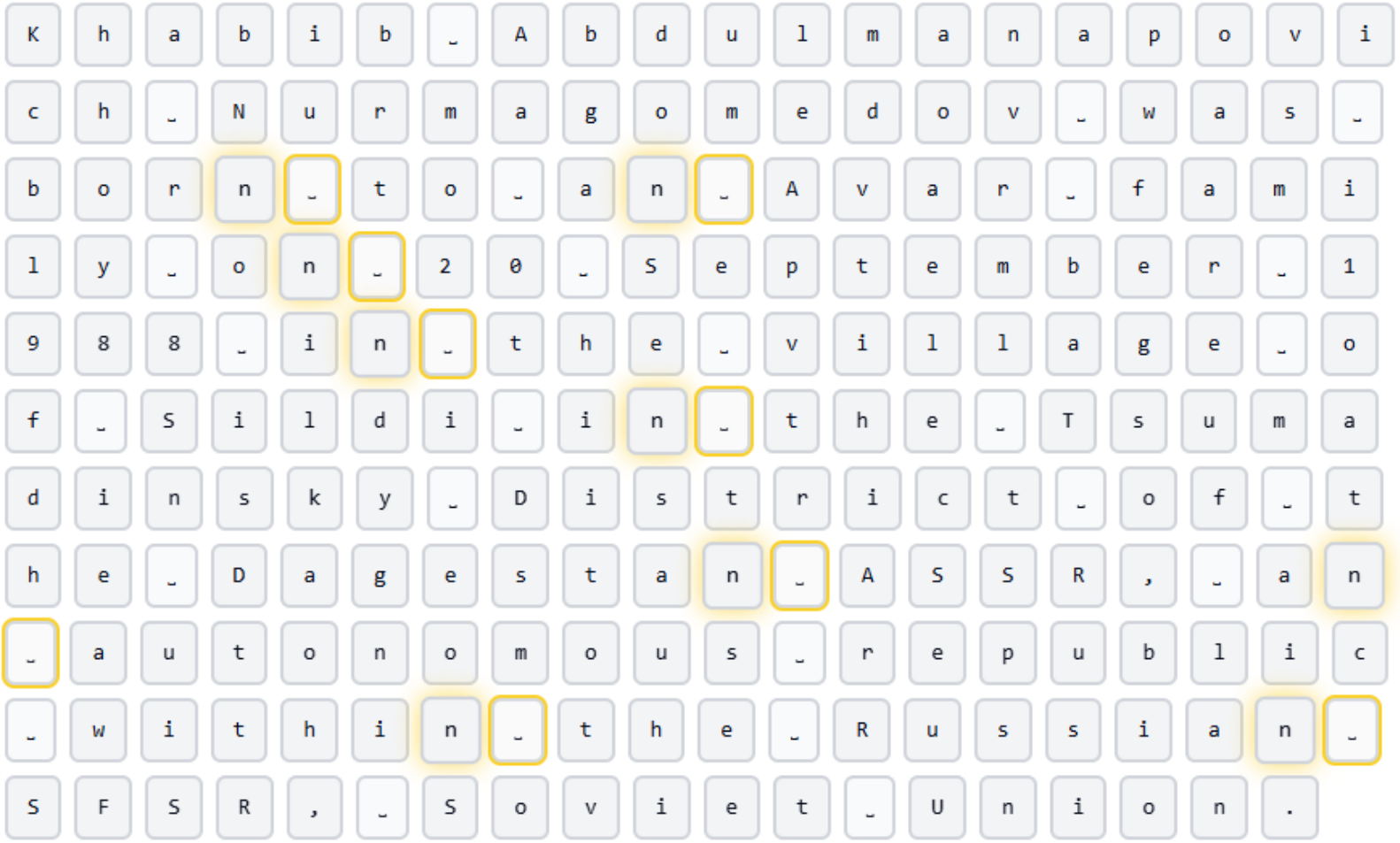
Merge Rules: Step 1 – Count pair frequencies

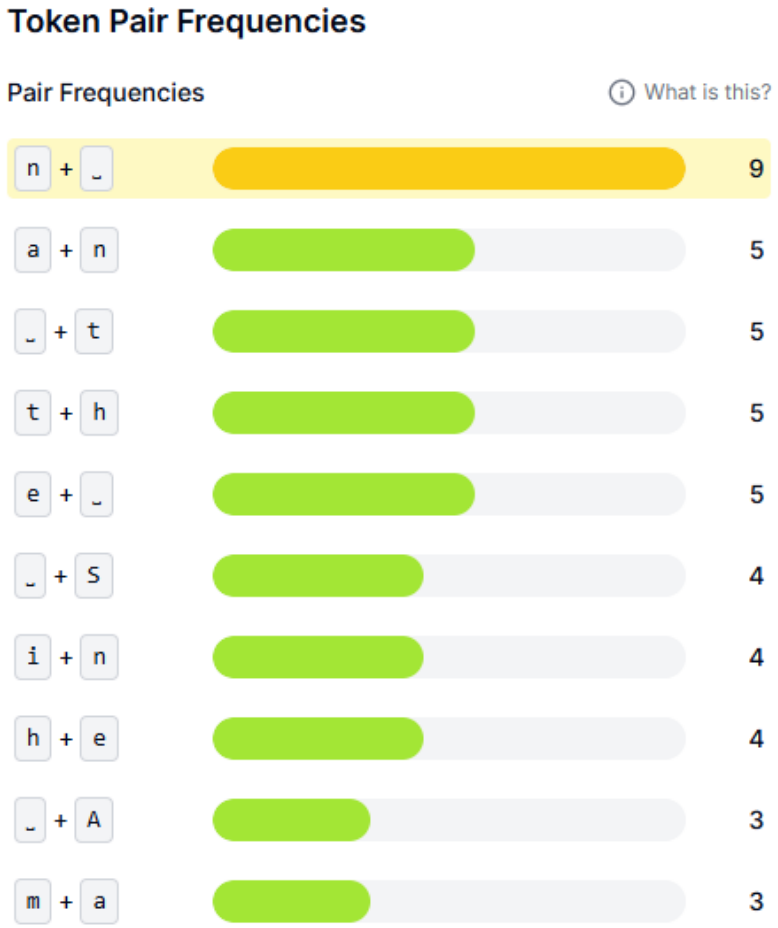




The most frequent pair is merged into a new token: n + _ → n_

Merge Rules: Step 1 – Count pair frequencies



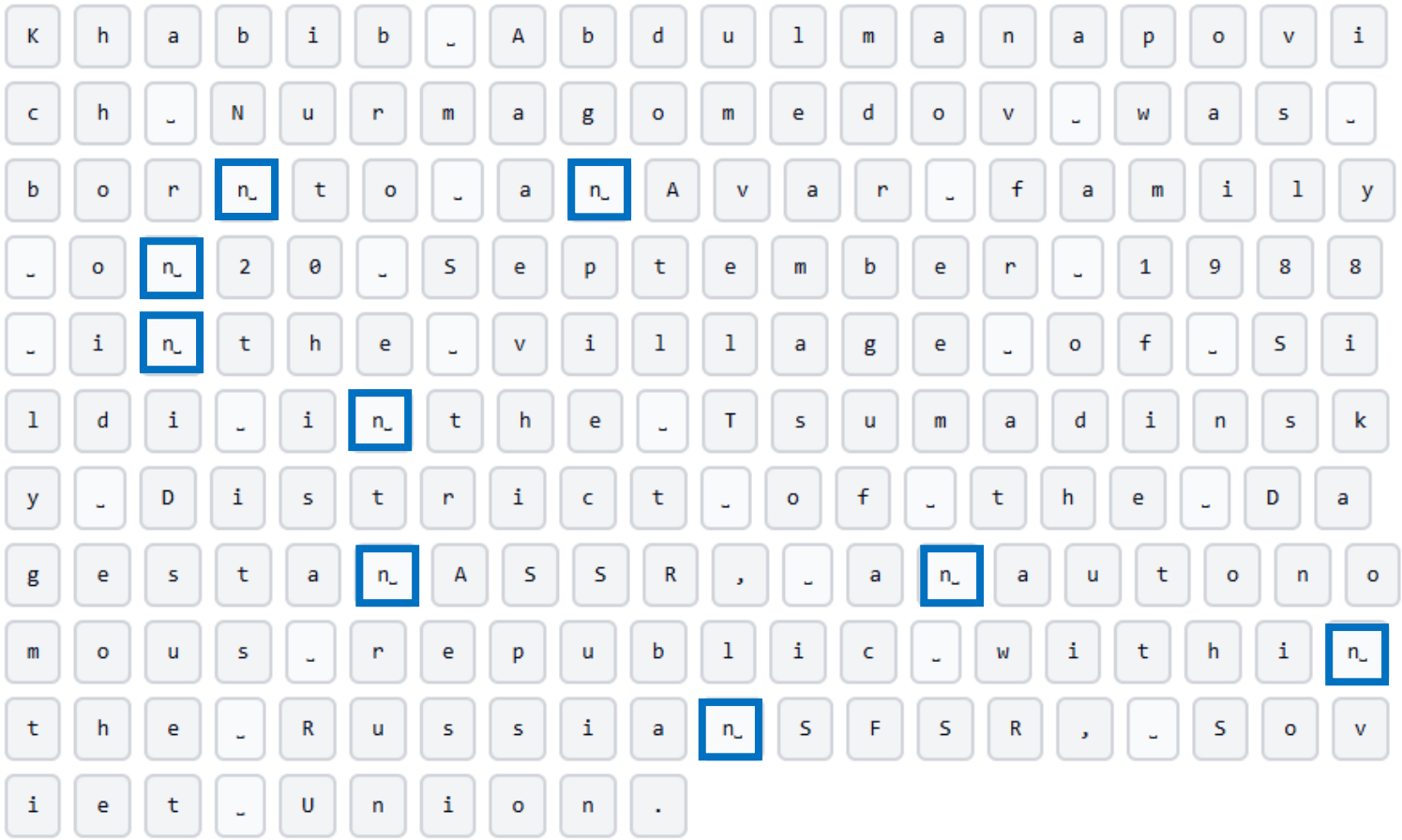


The most frequent pair is merged into a new token: n + _ → n_

Note that, in current tokenizers, we do not cross the space boundaries when forming merges

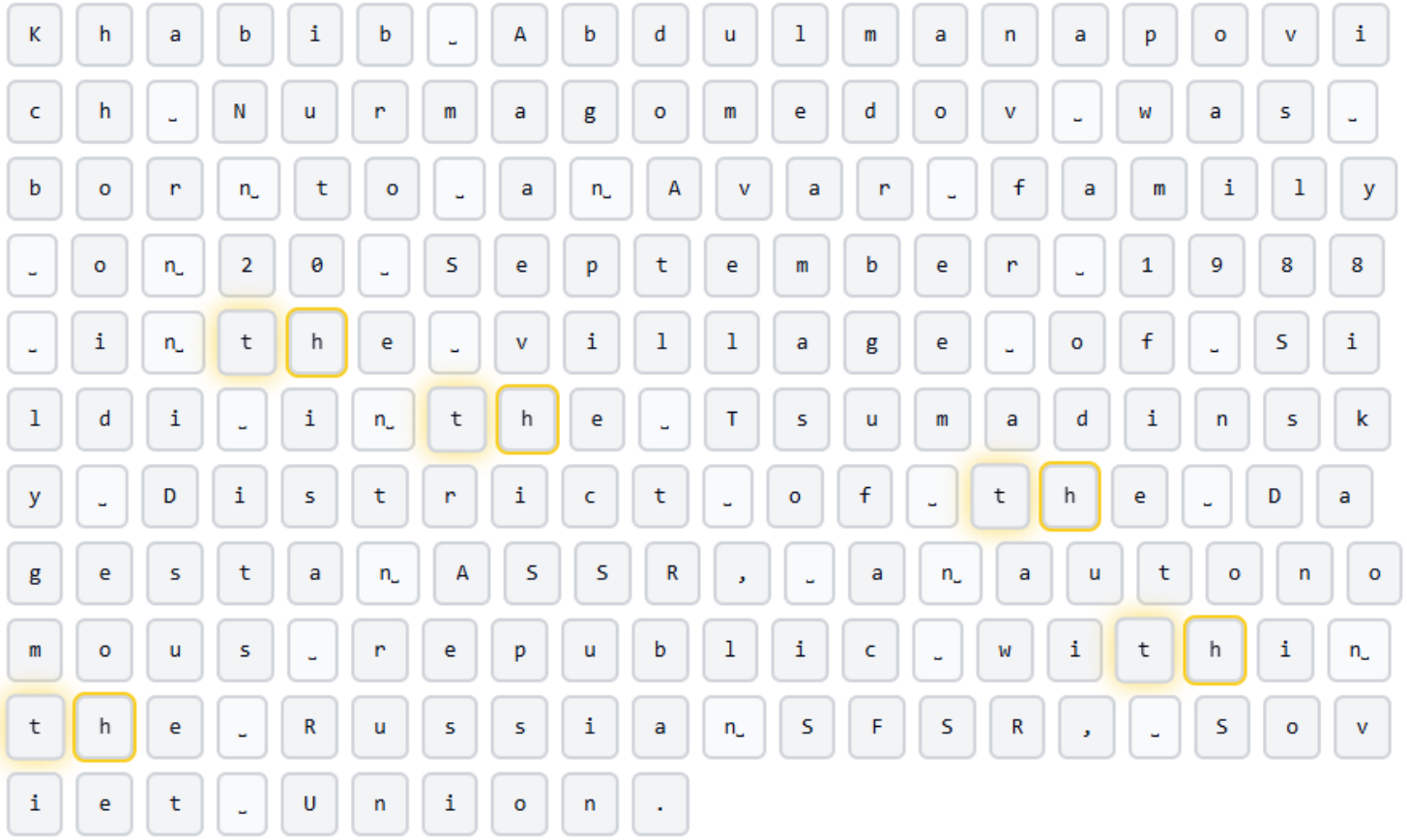
(Though in this visualization tool it does cross boundaries, as you can see from some pair counts)

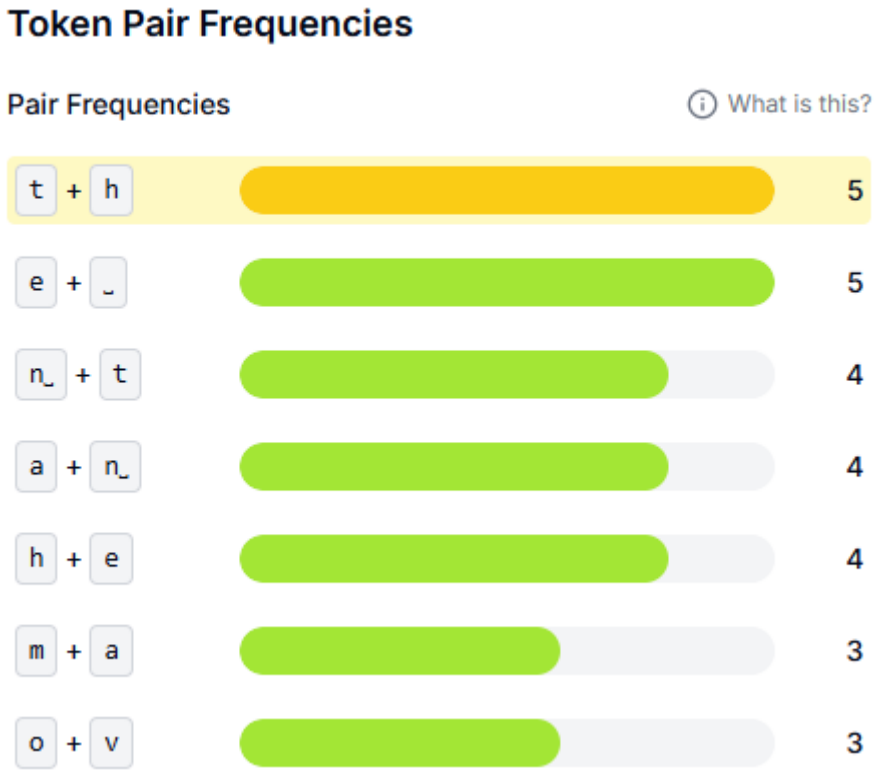
Merge Rules: Step 2 – Apply merge to corpus



The past merge is first applied to the corpus: n + _ → n_

Merge Rules: Step 3 – Count pair frequencies again





Recompute pair counts on corpus to identify new most frequent pair.

Add the identified pair to the vocabulary as a merge rule.

This is repeatedly done until we reach a desired vocabulary size K.

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b $\rightarrow$ ab
m e $\rightarrow$ me
…
h ab $\rightarrow$ hab
me d $\rightarrow$ med
…

Inference Algorithm: Apply Merge Rules to Tokenize

Khabib Nurmagomedov

Merge Rules

a
b
c
..
y
z
…
b_
…
a b $\rightarrow$ ab
m e $\rightarrow$ me
…
h ab $\rightarrow$ hab
me d $\rightarrow$ med
…

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

K h a b i b_ N u r m a g o m e d o v

Split by character if not

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

K h a b i b_ N u r m a g o m e d o v

Split by character if not

K h ab i b_ N u r m a g o me d o v

Apply merge rules in order

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

K h a b i b_ N u r m a g o m e d o v

Split by character if not

K h ab i b_ N u r m a g o me d o v

Apply merge rules in order

K hab i b_ N u r m a g o med o v

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

K h a b i b_ N u r m a g o m e d o v

Split by character if not

K h ab i b_ N u r m a g o me d o v

Apply merge rules in order

K hab i b_ N u r m a g o med o v

Tokenized text: *(14 tokens in this example)*

[‘K’, ‘hab’, ‘i’, ‘b_’, ‘N’, ‘u’, ‘r’, ‘m’, ‘a’, ‘g’, ‘o’, ‘med’, ‘o’, ‘v’]

Inference Algorithm: Apply Merge Rules to Tokenize

Merge Rules

a
b
c
..
y
z
…
b_
…
a b → ab
m e → me
…
h ab → hab
me d → med
…

Khabib Nurmagomedov

Khabib_ Nurmagomedov

First split on whitepace and see if word is already a token in vocab

K h a b i b_ N u r m a g o m e d o v

Split by character if not

K h ab i b_ N u r m a g o me d o v

Apply merge rules in order

K hab i b_ N u r m a g o med o v

Tokenized text: *(14 tokens in this example)*

[‘K’, ‘hab’, ‘i’, ‘b_’, ‘N’, ‘u’, ‘r’, ‘m’, ‘a’, ‘g’, ‘o’, ‘med’, ‘o’, ‘v’]

Compression rate = $\frac{18\ (chars)}{14\ (tokens)} = 1.285$

What’s the problem with applying BPE on strings?

What’s the problem with applying BPE on strings?

Poor handling of rare or unseen characters:

If some things don’t end up in the vocabulary because of low frequency, they will be tokenized as <unk> tokens

What’s the problem with applying BPE on strings?

Poor handling of rare or unseen characters:

If some things don’t end up in the vocabulary because of low frequency, they will be tokenized as <unk> tokens

Large Vocabularies:

Considering all writing scripts, the initialization step by itself will make the vocabulary tens of thousands long

What’s the problem with applying BPE on strings?

Poor handling of rare or unseen characters:

If some things don’t end up in the vocabulary because of low frequency, they will be tokenized as <unk> tokens

Large Vocabularies:

Considering all writing scripts, the initialization step by itself will make the vocabulary tens of thousands long

Other Issues: *String encoding mismatch at test time, worse multilingual support, etc.*

What’s the problem with applying BPE on strings?

Poor handling of rare or unseen characters:

If some things don’t end up in the vocabulary because of low frequency, they will be tokenized as <unk> tokens

Large Vocabularies:

Considering all writing scripts, the initialization step by itself will make the vocabulary tens of thousands long

Other Issues: *String encoding mismatch at test time, worse multilingual support, etc.*

How do we address this? → Byte-level BPE *(learn vocab and merges on byte representations)*

UTF-8 to byte representation

UTF-8: variable length encoding that uses 1 to 4 bytes per character depending on where it falls in the range

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz			
U+0080	U+07FF	110xxxyy	10yyzzzz		
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz

UTF-8 to byte representation

UTF-8: variable length encoding that uses 1 to 4 bytes per character depending on where it falls in the range

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz			
U+0080	U+07FF	110xxxyy	10yyzzzz		
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz

Using this encoding scheme, we can represent any character with a list of bytes:

Character	Codepoint (Hex)	Codepoint (Decimal)	Bytes
T	U+0054	84	[84]
ش	U+0634	1588	[216, 180]
𡨸	U+21A38	137784	[240, 161, 168, 184]

Why use raw bytes as base units instead of characters?

Eliminate out-of-vocabulary issues:

- The fixed 256-symbol set of raw bytes can encode any character, eliminating out-of-vocabulary issues.
- For example, if we face a character that is not in the tokenizer vocabulary, such as 𠀋
- In these cases, we do a “byte-fallback” where this character is tokenized as its raw bytes [240, 161, 168, 184]

Better compression:

- If you tokenize at the character level for multilingual data, you might need tens of thousands of characters to cover all scripts.
- Byte-level BPE can achieve a similar level of expressiveness with a much smaller base vocabulary (typically 256 bytes plus merges).

Why use raw bytes as base units instead of characters?

Eliminate out-of-vocabulary issues:

- The fixed 256-symbol set of raw bytes can encode any character, eliminating out-of-vocabulary issues.
- For example, if we face a character that is not in the tokenizer vocabulary, such as 𠜎
- In these cases, we do a “byte-fallback” where this character is tokenized as its raw bytes [240, 161, 168, 184]

Khabib Abdulmanapovich Nurmagomedov was born to an Avar family on 20 September 1988 in the village of Sildi in the Tsumadinsky District of the Dagestan ASSR, an autonomous republic within the Russian SFSR, Soviet Union.

Khabib Abdulmanapovich Nurmagomedov was born to an Avar family on 20 September 1988 in the village of Sildi in the Tsumadinsky District of the Dagestan ASSR, an autonomous republic within the Russian SFSR, Soviet Union.

```
[75, 104, 97, 98, 105, 98, 32, 65, 98, 100, 117, 108, 109, 97, 110, 97, 112, 111, 118, 105, 99, 104, 32, 78, 117, 114,
109, 97, 103, 111, 109, 101, 100, 111, 118, 32, 119, 97, 115, 32, 98, 111, 114, 110, 32, 116, 111, 32, 97, 110, 32,
65, 118, 97, 114, 32, 102, 97, 109, 105, 108, 121, 32, 111, 110, 32, 50, 48, 32, 83, 101, 112, 116, 101, 109, 98, 101,
114, 32, 49, 57, 56, 56, 32, 105, 110, 32, 116, 104, 101, 32, 118, 105, 108, 108, 97, 103, 101, 32, 111, 102, 32, 83,
105, 108, 100, 105, 32, 105, 110, 32, 116, 104, 101, 32, 84, 115, 117, 109, 97, 100, 105, 110, 115, 107, 121, 32, 68,
105, 115, 116, 114, 105, 99, 116, 32, 111, 102, 32, 116, 104, 101, 32, 68, 97, 103, 101, 115, 116, 97, 110, 32, 65,
83, 83, 82, 44, 32, 97, 110, 32, 97, 117, 116, 111, 110, 111, 109, 111, 117, 115, 32, 114, 101, 112, 117, 98, 108,
105, 99, 32, 119, 105, 116, 104, 105, 110, 32, 116, 104, 101, 32, 82, 117, 115, 115, 105, 97, 110, 32, 83, 70, 83, 82,
44, 32, 83, 111, 118, 105, 101, 116, 32, 85, 110, 105, 111, 110, 46]
```

Text will be represented as bytes, then the same BPE process is applied

In this case, both the string and byte representation have the same length (219), since 1 latin character corresponds to 1 byte in UTF-8

تتميز الثورة السورية بأنها أوَّل ثورة مكتملة الأركان منذ الربيع العربي، وأنها ثورة قادرة على إحداث تغييرات جذرية في بنية النظام، بما يوفر فرصًا حقيقية لحالة صعود نهضوي كبير

تتميز الثورة السورية بأنها أوَّل ثورة مكتملة الأركان منذ الربيع العربي، وأنها ثورة قادرة على إحداث تغييرات جذرية في بنية النظام، بما يوفر فرصًا حقيقية لحالة صعود نهضوي كبير

```
[216, 170, 216, 170, 217, 133, 217, 138, 216, 178, 32, 216, 167, 217, 132, 216, 171, 217, 136, 216, 177, 216, 169, 32,
216, 167, 217, 132, 216, 179, 217, 136, 216, 177, 217, 138, 216, 169, 32, 216, 168, 216, 163, 217, 134, 217, 135, 216,
167, 32, 216, 163, 217, 136, 217, 145, 217, 142, 217, 132, 32, 216, 171, 217, 136, 216, 177, 216, 169, 32, 217, 133,
217, 131, 216, 170, 217, 133, 217, 132, 216, 169, 32, 216, 167, 217, 132, 216, 163, 216, 177, 217, 131, 216, 167, 217,
134, 32, 217, 133, 217, 134, 216, 176, 32, 216, 167, 217, 132, 216, 177, 216, 168, 217, 138, 216, 185, 32, 216, 167,
217, 132, 216, 185, 216, 177, 216, 168, 217, 138, 216, 140, 32, 217, 136, 216, 163, 217, 134, 217, 135, 216, 167, 32,
216, 171, 217, 136, 216, 177, 216, 169, 32, 217, 130, 216, 167, 216, 175, 216, 177, 216, 169, 32, 216, 185, 217, 132,
217, 137, 32, 216, 165, 216, 173, 216, 175, 216, 167, 216, 171, 32, 216, 170, 216, 186, 217, 138, 217, 138, 216, 177,
216, 167, 216, 170, 32, 216, 172, 216, 176, 216, 177, 217, 138, 216, 169, 32, 217, 129, 217, 138, 32, 216, 168, 217,
134, 217, 138, 216, 169, 32, 216, 167, 217, 132, 217, 134, 216, 184, 216, 167, 217, 133, 216, 140, 32, 216, 168, 217,
133, 216, 167, 32, 217, 138, 217, 136, 217, 129, 216, 177, 32, 217, 129, 216, 177, 216, 181, 217, 139, 216, 167, 32,
216, 173, 217, 130, 217, 138, 217, 130, 217, 138, 216, 169, 32, 217, 132, 216, 173, 216, 167, 217, 132, 216, 169, 32,
216, 181, 216, 185, 217, 136, 216, 175, 32, 217, 134, 217, 135, 216, 182, 217, 136, 217, 138, 32, 217, 131, 216, 168,
217, 138, 216, 177, 46]
```

In this Arabic text example, 1 character typically corresponds to 2 bytes

String representation length: 173 characters

Byte representation length: 317 bytes

Non-English and specifically non-latin script languages will typically be fragmented into more tokens, just because their order in the UTF-8 code point range is higher

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Merge list

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Merge list

Corpus (bytes)

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Merge list

Corpus (bytes)

Tokenization function (map bytes to tokens)

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Merge list

Corpus (bytes)

Tokenization function (map bytes to tokens)

Where: $\mathrm{CR}(\mathcal{D}; \tau) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{b} \in \mathcal{D}} \frac{|\mathbf{b}|_u}{|\tau(\mathbf{b})|}$

Byte string of a document

Formalization of BPE: Maximizing compression

BPE seeks the merge list (subject to a size constraint K) that maximizes the compression rate of the corpus:

$$\mathbf{m}^* = \max_{\mathbf{m}:|\mathbf{m}|=K} \mathrm{CR}(\mathcal{D}; \tau_{\mathbf{m}})$$

Merge list

Corpus (bytes)

Tokenization function (map bytes to tokens)

Where: $\mathrm{CR}(\mathcal{D}; \tau) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{b} \in \mathcal{D}} \frac{|\mathbf{b}|_u}{|\tau(\mathbf{b})|}$

Byte string of a document

BPE takes a greedy approach to choosing m, finding an approximate solution.

The subword-type pair with the highest count, is deemed the most “compressive”

Parity-aware Byte Pair Encoding

Vanilla BPE $\rightarrow$ chooses merges that maximize a global frequency objective, implicitly favoring the compression of languages with a larger presence in the training corpus.

Parity-aware Byte Pair Encoding

Vanilla BPE $\rightarrow$ chooses merges that maximize a global frequency objective, implicitly favoring the compression of languages with a larger presence in the training corpus.

Parity-aware BPE $\rightarrow$ replaces this global objective with a max–min criterion: at every step, it selects the merge that most improves the language that currently has the smallest compression rate.

$$\mathbf{m}^\star = \max_{\mathbf{m}:|\mathbf{m}|=K} \min_{\ell} \mathrm{CR}(\ell; \tau_{\mathbf{m}})$$

Parity-aware Byte Pair Encoding

Vanilla BPE $\rightarrow$ chooses merges that maximize a global frequency objective, implicitly favoring the compression of languages with a larger presence in the training corpus.

Parity-aware BPE $\rightarrow$ replaces this global objective with a max–min criterion: at every step, it selects the merge that most improves the language that currently has the smallest compression rate.

$$\mathbf{m}^\star = \max_{\mathbf{m}:|\mathbf{m}|=K} \min_{\ell} \mathrm{CR}(\ell; \tau_{\mathbf{m}})$$

At merge step *k*, it identifies the language with the worst compression under the tokenizer defined by the merge list thus far:

$$\ell^\star = \arg\min_{\ell \in \mathcal{L}} \mathrm{CR}(\ell; \tau_{\mathbf{m}_{<k}})$$

Parity-aware Byte Pair Encoding

Vanilla BPE → chooses merges that maximize a global frequency objective, implicitly favoring the compression of languages with a larger presence in the training corpus.

Parity-aware BPE → replaces this global objective with a max–min criterion: at every step, it selects the merge that most improves the language that currently has the smallest compression rate.

$$\mathbf{m}^\star = \max_{\mathbf{m}:|\mathbf{m}|=K} \min_{\ell} \mathrm{CR}(\ell; \tau_{\mathbf{m}})$$

At merge step *k*, it identifies the language with the worst compression under the tokenizer defined by the merge list thus far:

$$\ell^\star = \arg\min_{\ell \in \mathcal{L}} \mathrm{CR}(\ell; \tau_{\mathbf{m}_{<k}})$$

The compression rate between languages is compared on a parallel corpus with aligned segments (FLORES)

Once this is identified, the most frequent pairs from the corpus of this language is chosen to create a merge

Parity-aware Byte Pair Encoding: Variants

Hybrid parity-aware BPE $\rightarrow$ uses the global objective of Classical BPE for the first K merges, then switches to the parity-aware objective for another J merges

Parity-aware Byte Pair Encoding: Variants

Hybrid parity-aware BPE $\rightarrow$ uses the global objective of Classical BPE for the first K merges, then switches to the parity-aware objective for another J merges

Moving-window balancing $\rightarrow$ There may be a point where the compression in a language no longer or barely improves, even if it is repeatedly chosen for the next merge.

To prevent the algorithm from being “stuck” selecting the same language exclusively, track the W most recent languages, and do not select a language if it occurs more than $\alpha \frac{W}{|L|}$ times in this moving window

Experimental Setup

Tokenizer Training Data: mC4 corpus

Model: 3B transformer decoder model based on llama architecture

Model Training Data: Fineweb2 (100B tokens)

Hardware: 256 GPUs (64x4 H100s) - takes around 18h per 100B tokens

Intrinsic Eval *(task-independent tokenizer properties)*

These metrics encompass basic tokenization properties, information-theoretic measures, cross-linguistic fairness, and morphological alignment

Extrinsic Eval *(downstream model performance trained using the tokenizer)*

10 multilingual benchmarks (Belebele, mTruthfulQA, PAWS-X, XNLI, Xcodah, XCSQA, XStoryCloze, Xwinogrande, MMMLU, INCLUDE, Exams, M3Exams)

Intrinsic Metrics

Fertility: measures the average tokens a word is broken up into.

Compression Rate: measures the degree to which a unit of text (document) has been shrunk after applying the given tokenizer (higher is more compression).

Vocabulary Utilization: is the fraction of the tokenizer's vocabulary that appears in the evaluation corpus. Low utilization for a language signals wasted capacity or—when there are large differences across languages—biased vocabulary allocation.

Tokenizer Fairness Gini: adapts the Gini coefficient to the per-language tokenization cost distribution (e.g., tokens per document in a parallel corpus). Values near 0 mean equal cost across languages; values closer to 1 indicate inequality

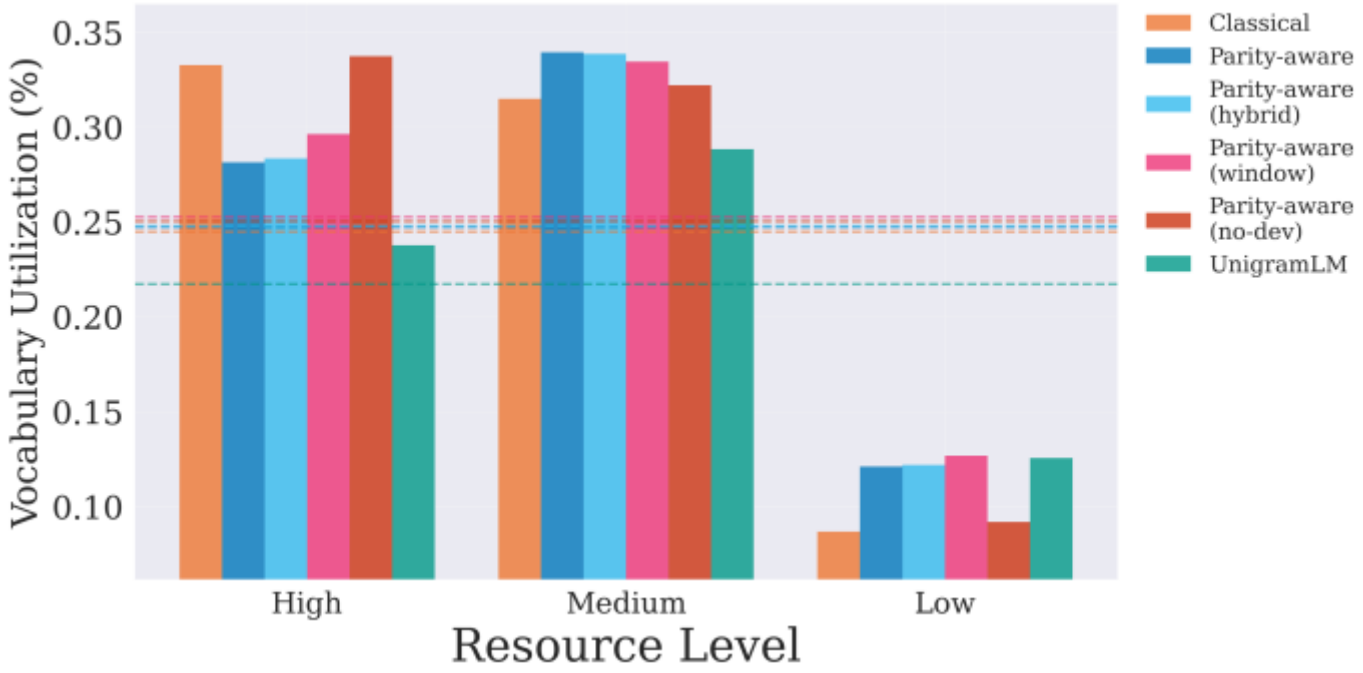
MorphScore: measures how well token boundaries align with true morpheme boundaries, computed as morpheme-level precision/recall (and F1). High scores mean tokens respect morphological structure; low precision implies over-segmentation, while low recall may suggest under-segmentation

Type–Token Ratio: diversity - whether the vocabulary is dominated by repeated tokens or contains many unique ones.

Rényi entropy: distributional concentration - whether a few tokens dominate usage or if frequencies are more evenly spread

https://github.com/cimeister/tokenizer-analysis-suite

Vocabulary Utilization



For low- and medium-resource languages, Parity BPE variants achieve higher vocabulary utilization than BPE.

This highlights their effectiveness at providing fairer vocabulary allocation across languages.

Intrinsic Evaluation Results

Tokenizer	Type-Token Ratio	Fertility	Compression Rate	Rényi Entropy (α=2.5)	Gini Coefficient	MorphScore Precision	MorphScore Recall
Classical	0.0743	4.260 ± 0.049	0.0303 ± 0.0001	**8.13**	0.064	0.412 ± 0.051	0.456 ± 0.049
UnigramLM	0.0475	4.612 ± 0.042	0.0228 ± 0.0001	4.68	0.094	0.153 ± 0.037	0.268 ± 0.053
Parity-aware	0.0765	4.204 ± 0.049	0.0300 ± 0.0001	8.12	**0.011**	0.407 ± 0.051	0.457 ± 0.049
Parity-aware (hybrid)	0.0770	**4.191 ± 0.049**	0.0303 ± 0.0001	8.10	0.018	0.412 ± 0.051	0.457 ± 0.049
Parity-aware (window)	0.0788	4.219 ± 0.050	0.0302 ± 0.0001	8.11	0.013	0.405 ± 0.049	0.453 ± 0.047
Parity-aware (window+hybrid)	**0.0794**	4.203 ± 0.050	**0.0305 ± 0.0001**	8.09	0.022	0.416 ± 0.049	0.460 ± 0.047
Parity-aware (no-dev)	0.0772	4.310 ± 0.050	0.0303 ± 0.0001	8.12	0.059	**0.423 ± 0.051**	**0.466 ± 0.049**

Parityaware BPE outperform Classical BPE in terms of the Gini coefficient, indicating more equitable token costs per document across languages

Classical BPE and the Parity-aware BPE variants attain almost identical compression and Rényi entropies; evidence that the parity-aware variants match global efficiency while redistributing it more evenly.

Extrinsic Evaluation Results

Language	Classical BPE	Parity-aware (hybrid)	Parity-aware (window+hybrid)	Random
Arabic	38.19 ± 2.90	**39.04** ± 2.89	38.84 ± 2.90	32.00
Bengali	24.95 ± 3.09	23.54 ± 2.98	23.91 ± 3.01	**25.00**
German	32.92 ± 3.14	34.78 ± 3.66	**36.82** ± 4.04	30.62
Greek	41.95 ± 3.18	42.55 ± 3.22	**43.16** ± 3.25	37.50
Spanish	37.53 ± 2.66	38.83 ± 2.71	**39.30** ± 2.75	32.77
Persian	**42.80** ± 5.39	39.15 ± 5.27	39.15 ± 5.27	25.00
French	**38.67** ± 3.90	36.59 ± 2.84	37.10 ± 2.82	32.00
Hindi	**33.92** ± 2.25	33.92 ± 2.24	33.86 ± 2.24	30.62
Indonesian	38.95 ± 2.62	**40.55** ± 2.66	40.46 ± 2.66	35.00
Italian	32.82 ± 2.86	**35.01** ± 3.00	34.62 ± 2.98	27.22
Japanese	37.43 ± 2.39	**37.45** ± 2.39	37.43 ± 2.39	34.00
Korean	33.00 ± 5.22	33.00 ± 5.22	**34.33** ± 5.29	25.00
Polish	29.75 ± 2.50	**31.14** ± 2.60	28.97 ± 2.49	23.75
Portuguese	**33.63** ± 2.81	33.15 ± 2.77	33.06 ± 2.77	27.50
Russian	36.36 ± 2.27	36.21 ± 2.26	**36.57** ± 2.28	32.77
Tamil	31.32 ± 2.81	**32.25** ± 2.90	32.19 ± 2.90	31.25
Telugu	32.73 ± 2.61	**33.52** ± 2.61	33.26 ± 2.61	30.00
Turkish	**39.04** ± 2.89	38.46 ± 2.83	37.89 ± 2.75	35.00
Vietnamese	33.69 ± 2.31	**33.87** ± 2.27	33.80 ± 2.30	29.50
Chinese	38.43 ± 2.11	**38.58** ± 2.11	38.32 ± 2.10	35.00
English	43.04 ± 1.84	**44.15** ± 1.85	43.74 ± 1.85	35.50
Thai	40.76 ± 1.62	40.96 ± 1.63	**41.06** ± 1.63	37.50

Average accuracy across 13 benchmarks

Parity-aware BPE maintains performance across languages: accuracy changes relative to Classical BPE are small.

Parity-aware tokenizers can more fairly tokenize diverse languages without compromising LM performance.