Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

A Formal Perspective on Byte-Pair Encoding

formalization & bounds..

Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

A Formal Perspective on Byte-Pair Encoding

formalization & bounds..

Tokenization and the Noiseless Channel

how to apriori choose the best tokenization and how we published a wrong paper..

Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

A Formal Perspective on Byte-Pair Encoding

formalization & bounds..

Tokenization and the Noiseless Channel

how to apriori choose the best tokenization and how we published a wrong paper..

Distributional Properties of Subword Regularization

and we use tokenization incorrectly..

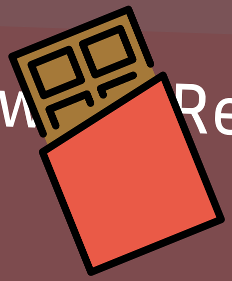
Token(s) of Appreciation for BPE

Vilém Zouhar et al, September 2024

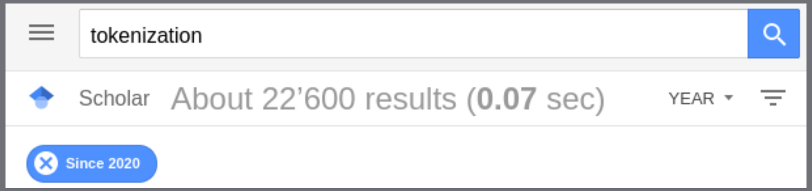
A Formal Perspective on Byte-Pair Encoding
formalization & bounds..

Tokenization and the Noiseless Channel
how to apriori choose the best tokenization and how we published a wrong paper..

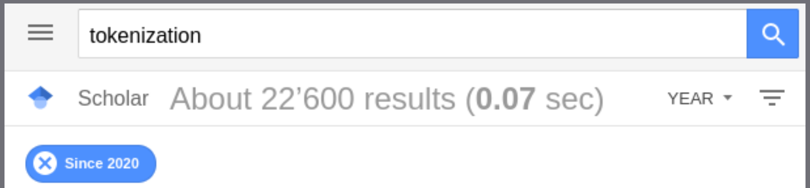
Distributional Properties of Subw Re
and we use tokenization incorrectly..

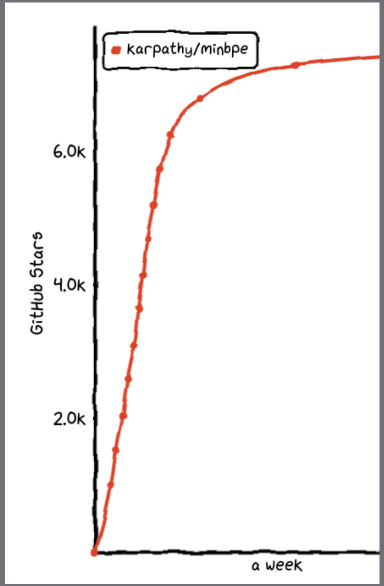


Trick question for a chocolate at the end.




```
vilda@karotte:~$ grep -i "token\|BPE\|byte.pair" acl-anthology.bib |  wc  -l
3109
vilda@karotte:~$
```





```
vilda@karotte:~$ grep -i "token\|BPE\|byte.pair" acl-anthology.bib |  wc  -l
3109
vilda@karotte:~$ 
```

How to represent text?

“he loved pickled pickles”

How to represent text?

“he loved pickled pickles”

[h, e, _, l, o, v, e, d, _, p, i, c, k, l, e, d, _, p, i, c, k, l, e, s]

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

How to represent text?

“he loved pickled pickles”

[h, e, _, l, o, v, e, d, _, p, i,

meaningless items, long sequences

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

How to represent text?

"he loved pickled pickles"

[h, e, _, l, o, v, e, d, _, p, i,

meaningless items, long sequences

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

[he, _, loved, _, pickled, _, pickles]

[1045, 0, 30123, 0, 6232, 0, 72057]

How to represent text?

"he loved pickled pickles"

[h, e, _, l, o, v, e, d, _, p, i,

meaningless items, long sequences

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

[he, _, loved, _, pickled, _

high dimensionality, rare & unknown words

[1045, 0, 30123, 0, 6232, 0, 72057]

How to represent text?

"he loved pickled pickles"

[h, e, _, l, o, v, e, d, _, p, i,

meaningless items, long sequences

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

[he, _, loved, _, pickled, _

high dimensionality, rare & unknown words

[1045, 0, 30123, 0, 6232, 0, 72057]

[he, _, lov, ed, _, pickl, ed, _, pickl, es]

[532, 0, 20, 952, 0, 1911, 952, 0, 1911, 12]

How to represent text?

"he loved pickled pickles"

[h, e, _, l, o, v, e, d, _, p, i,

[104, 101, 32, 108, 111, 118, 101, 100, 32, 112, 105, 99, 107, 108, 101, 100, 32, 112, 105, 99, 107, 108, 101, 115]

meaningless items, long sequences

[he, _, loved, _, pickled, _

[1045, 0, 30123, 0, 6232, 0, 72057]

high dimensionality, rare & unknown words

[he, _, lov, ed, _, pickl, e

[532, 0, 20, 952, 0, 1911, 952, 0, 1911, 12]

how to find those?

Finding meaningful *subwords*

Morphology segmentation:

- [`he, _, lov, ed, _, pickl, ed, _, pickl, es`]

Finding meaningful *subwords*

Morphology segmentation:

- [`he, _, lov, ed, _, pickl, ed, _, pickl, es`]
- requires language knowledge
- what if we desire higher/lower granularity?

Finding meaningful *subwords*

Morphology segmentation:

- [`he, _, lov, ed, _, pickl, ed, _, pickl, es`]
- requires language knowledge
- what if we desire higher/lower granularity?

Dedicated algorithm (wish):

- unsupervised
- granularity/vocabulary size hyperparameter
  - [`he, _, loved, _, pickl, ed, _, pickl, es`]
  - [`he, _, lov, ed, _, pic, kl, ed, _, pic, kl, es`]

Finding meaningful *subwords*

Morphology segmentation:

- `[he, _, lov, ed, _, pickl, ed, _, pickl, es]`
- requires language knowledge
- what if we desire higher/lower granularity?

Byte-Pair Encoding:

- unsupervised
- granularity/vocabulary size hyperparameter
  - `[he, _, loved, _, pickl, ed, _, pickl, es]`
  - `[he, _, lov, ed, _, pic, kl, ed, _, pic, kl, es]`

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```

Byte-Pair Encoding

`t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s`

merge `p i` → `pi`

Byte-Pair Encoding

t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

merge p i → pi

t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s

Byte-Pair Encoding

t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

merge p i → pi

t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s

merge c k → ck

Byte-Pair Encoding

t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

merge p i → pi

t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s

merge c k → ck

t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s

Byte-Pair Encoding

t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

merge p i → pi

t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s

merge c k → ck

t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s

merge pi ck → pick

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```
merge `p i` → `pi`
```
t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s
```
merge `c k` → `ck`
```
t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s
```
merge `pi ck` → `pick`
```
t h e y _ pick e d_ pick l e d _ pick l e s
```

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```
merge `p i` → `pi`
```
t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s
```
merge `c k` → `ck`
```
t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s
```
merge `pi ck` → `pick`
```
t h e y _ pick e d_ pick l e d _ pick l e s
```
merge `pick l` → `pickl`

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```
merge `p i` → `pi`
```
t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s
```
merge `c k` → `ck`
```
t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s
```
merge `pi ck` → `pick`
```
t h e y _ pick e d_ pick l e d _ pick l e s
```
merge `pick l` → `pickl`
```
t h e y _ pick e d_ pickl e d _ pickl e s
```

Byte-Pair Encoding

t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

merge p i → pi

t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s

merge c k → ck

t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s

merge pi ck → pick

t h e y _ pick e d_ pick l e d _ pick l e s

merge pick l → pickl

t h e y _ pick e d_ pickl e d _ pickl e s

Unsupervised ?

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```

merge `p i` → `pi`

```
t h e y _ pi c k e d _  pi c k l e d _  pi c k l e s
```

merge `c k` → `ck`

```
t h e y _ pi ck e d_    pi ck l e d _   pi ck l e s
```

merge `pi ck` → `pick`

```
t h e y _ pick e d_     pick l e d _    pick l e s
```

merge `pick l` → `pickl`

```
t h e y _ pick e d_     pickl e d _     pickl e s
```

Unsupervised ✓   Granularity ?

Byte-Pair Encoding

```
t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s
```
merge `p i` → `pi`
```
t h e y _ pi c k e d _ pi c k l e d _ pi c k l e s
```
merge `c k` → `ck`
```
t h e y _ pi ck e d_ pi ck l e d _ pi ck l e s
```
merge `pi ck` → `pick`
```
t h e y _ pick e d_ pick l e d _ pick l e s
```
merge `pick l` → `pickl`
```
t h e y _ pick e d_ pickl e d _ pickl e s
```

Unsupervised ✓ Granularity ✓

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
6. **end for**

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, \ldots, M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

Output example:

$\overline{x}$ = `t h e y _ pick ed _ pickl ed _ pickl e s`

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \, \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

Runtime:

$O(M) \times$?

Output example:

$\overline{x}$ = `t h e y _ pick ed _ pickl ed _ pickl e s`

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

Runtime:

$$O(M) \times (\mathrm{argmax}\ \mathrm{Freq} + \mathrm{ApplyMerge})$$

Output example:

$\overline{x}$ = `t h e y _ pick ed _ pickl ed _ pickl e s`

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \ \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

Runtime:

$$O(M \times |\overline{x}|)$$

Output example:

$\overline{x}$ = `t h e y _ pick ed _ pickl ed _ pickl e s`

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩

Byte-Pair Encoding

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \ \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

Runtime:

$O(M \times |\overline{x}|)$

Not good, will fix later

Output example:

$\overline{x}$ = `t h e y _ pick ed _ pickl ed _ pickl e s`

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩

BPE is Compression

$s_1 =$ t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s

$|s_1| = 27$

$|V_1| = 12$

BPE is Compression

$s_1 = $ `t h e y _ p i c k e d _ p i c k l e d _ p i c k l e s`

$|s_1| = 27$

$|V_1| = 12$

cost $= |s_1| \cdot \lceil \log_2 |V_1| \rceil = 22 \cdot 4 = 108$ bits

BPE is Compression

$s_1 = \texttt{t h e y \_ p i c k e d \_ p i c k l e d \_ p i c k l e s}$

$|s_1| = 27$

$|V_1| = 12$

$\text{cost} = |s_1| \cdot \lceil \log_2 |V_1| \rceil = 22 \cdot 4 = 108 \text{ bits}$

$s_2 = \texttt{t h e y \_ pick ed \_ pickl ed \_ pickl e s}$

$|s_2| = 14$

$|V_2| = |V_1| + |\{\texttt{pi}, \texttt{ck}, \texttt{pick}, \texttt{pickl}, \texttt{ed}\}| = 17$

BPE is Compression

$s_1 = \texttt{t h e y \_ p i c k e d \_ p i c k l e d \_ p i c k l e s}$

$|s_1| = 27$

$|V_1| = 12$

$\text{cost} = |s_1| \cdot \lceil \log_2 |V_1| \rceil = 22 \cdot 4 = 108 \text{ bits}$

$s_2 = \texttt{t h e y \_ pick ed \_ pickl ed \_ pickl e s}$

$|s_2| = 14$

$|V_2| = |V_1| + |\{\texttt{pi}, \texttt{ck}, \texttt{pick}, \texttt{pickl}, \texttt{ed}\}| = 17$

$\text{cost} = |s_2| \cdot \lceil \log_2 |V_2| \rceil = 14 \cdot 5 = 70 \text{ bits}$

BPE is Greedy

BPE is Greedy

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

BPE is Greedy

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
6. **end for**

BPE is Greedy

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad\mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad\overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad\overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
6. **end for**

Decision example:

$\overline{x}$ = a b a a b b a a

$\mathrm{Freq}(x, (\mu', \mu''))$ = ⟨ a b → 2, b a → 2, a a → 2, b b → 1 ⟩

BPE is *not* Optimal

$$\begin{aligned}\overline{x} &= \texttt{a b a a b b a a} \\ &= \texttt{ab a ab b a a} \\ &= \texttt{aba ab b a a}\end{aligned}$$

BPE is *not* Optimal

$$\overline{x} = \texttt{a b a a b b a a}$$
$$= \texttt{ab a ab b a a}$$
$$= \texttt{aba ab b a a}$$

$$\overline{\mu}^{\blacktriangle} = \langle \texttt{a b} \to \texttt{ab}, \quad \texttt{ab a} \to \texttt{aba} \rangle$$

$$|\text{ApplyMerges}(\overline{\mu}^{\blacktriangle}, \overline{x})| = 5$$

BPE is *not* Optimal

$$\overline{x} = \texttt{a b a a b b a a}$$
$$= \texttt{ab a ab b a a}$$
$$= \texttt{aba ab b a a}$$

$\overline{\mu}^{\blacktriangle} = \langle \texttt{a b} \rightarrow \texttt{ab}, \quad \texttt{ab a} \rightarrow \texttt{aba} \rangle$ $\qquad |\mathrm{ApplyMerges}(\overline{\mu}^{\blacktriangle}, \overline{x})| = 5$

$$\overline{x} = \texttt{a b a a b b a a}$$
$$= \texttt{a ba a b ba a}$$
$$= \texttt{a baa b baa}$$

$\overline{\mu}^{\star} = \langle \texttt{b a} \rightarrow \texttt{ba}, \quad \texttt{ba a} \rightarrow \texttt{baa} \rangle$

$\mathrm{ApplyMerges}(\overline{\mu}^{\star}, \overline{x}) = \texttt{a baa b baa}$ $\qquad |\mathrm{ApplyMerges}(\overline{\mu}^{\star}, \overline{x})| = 4$

BPE is *Approximately*-Optimal

Given any $\overline{x} \in \Sigma^*$

$c^{\blacktriangle} =$ compression length with BPE $\quad |\mathrm{BPE}(\overline{x})_0|$

$c^{\star} =$ best compression length $\quad \min_{\overline{\mu},\ |\overline{\mu}| \leq M} |\mathrm{ApplyMerges}(\overline{\mu}, \overline{x})|$

$|\overline{x}| - c^{\blacktriangle} =$ how many characters were saved $\quad$ (higher is better)

BPE is *Approximately*-Optimal

Given any $\overline{x} \in \Sigma^*$

$c^{\blacktriangle} =$ compression length with BPE $\quad |\mathrm{BPE}(\overline{x})_0|$

$c^{\star} =$ best compression length $\quad \min_{\overline{\mu},\ |\overline{\mu}| \leq M} |\mathrm{ApplyMerges}(\overline{\mu}, \overline{x})|$

$|\overline{x}| - c^{\blacktriangle} =$ how many characters were saved $\quad$ (higher is better)

$$\frac{|\overline{x}| - c^{\blacktriangle}}{|\overline{x}| - c^{\star}} \geq \frac{1 - e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})} \approx 37\%$$

BPE is *Approximately*-Optimal

Given any $\overline{x} \in \Sigma^*$

$c^{\blacktriangle}$ = compression length with BPE $\quad |\mathrm{BPE}(\overline{x})_0|$

$c^{\star}$ = best compression length $\quad \min_{\overline{\mu},\ |\overline{\mu}| \leq M} |\mathrm{ApplyMerges}(\overline{\mu}, \overline{x})|$

$|\overline{x}| - c^{\blacktriangle}$ = how many characters were saved (higher is better)

$$\frac{|\overline{x}| - c^{\blacktriangle}}{|\overline{x}| - c^{\star}} \geq \frac{1 - e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})} \approx 37\%$$

Paper: A Formal Perspective on Byte-Pair Encoding

BPE is *Approximately*-Optimal: Proof

Let $\kappa(\overline{\mu}) = |x| - c^{\blacktriangle}$

We get $\frac{|\overline{x}| - c^{\blacktriangle}}{|\overline{x}| - c^{\star}} \geq \frac{1 - e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})}$ if $\kappa(\overline{\mu})$ is:

BPE is *Approximately*-Optimal: Proof

Let $\kappa(\overline{\mu}) = |x| - c^{\blacktriangle}$

We get $\frac{|\overline{x}|-c^{\blacktriangle}}{|\overline{x}|-c^{\star}} \geq \frac{1-e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})}$ if $\kappa(\overline{\mu})$ is:

(1) monotone non-decreasing, and

(2) submodular, and

(3) hiearchical sequence-submodular.

BPE is *Approximately*-Optimal: Proof

Let $\kappa(\overline{\mu}) = |x| - c^{\blacktriangle}$

We get $\frac{|\overline{x}| - c^{\blacktriangle}}{|\overline{x}| - c^{\star}} \geq \frac{1 - e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})}$ if $\kappa(\overline{\mu})$ is:

(1) monotone non-decreasing, and
(2) submodular, and
(3) hiearchical sequence-submodular.

(1) monotone non-decreasing: $\kappa(\mu\nu) \geq \kappa(\mu)$

BPE is *Approximately*-Optimal: Proof

Let $\kappa(\overline{\mu}) = |x| - c^{\blacktriangle}$

We get $\frac{|\overline{x}|-c^{\blacktriangle}}{|\overline{x}|-c^{\star}} \geq \frac{1-e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})}$ if $\kappa(\overline{\mu})$ is:

(1) monotone non-decreasing, and
(2) submodular, and
(3) hiearchical sequence-submodular.

(1) monotone non-decreasing: $\kappa(\mu\nu) \geq \kappa(\mu)$

(2) submodular: $\kappa(\nu \mid \mu) \geq \kappa(\nu \mid \mu\omega) \Leftrightarrow \kappa(\mu\nu) - \kappa(\mu) \geq \kappa(\mu\omega\nu) - \kappa(\mu\omega)$

BPE is *Approximately*-Optimal: Proof

Let $\kappa(\overline{\mu}) = |x| - c^{\blacktriangle}$

We get $\frac{|\overline{x}| - c^{\blacktriangle}}{|\overline{x}| - c^{\star}} \geq \frac{1 - e^{-\sigma(\overline{\mu}^{\star})}}{\sigma(\overline{\mu}^{\star})}$ if $\kappa(\overline{\mu})$ is:

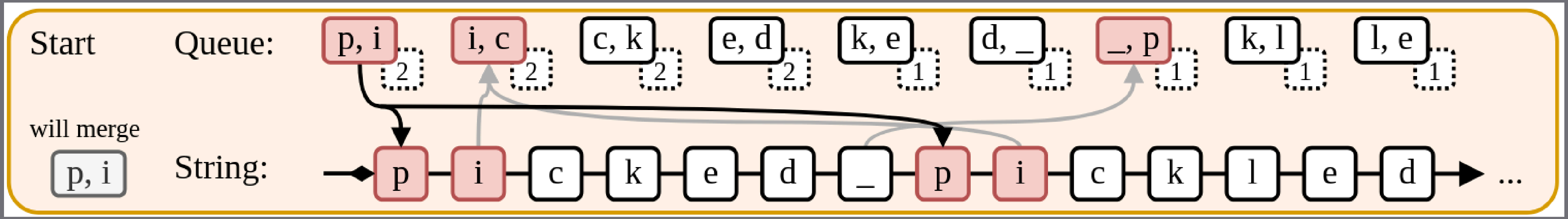
(1) monotone non-decreasing, and
(2) submodular, and
(3) hiearchical sequence-submodular.

(1) monotone non-decreasing: $\kappa(\mu\nu) \geq \kappa(\mu)$

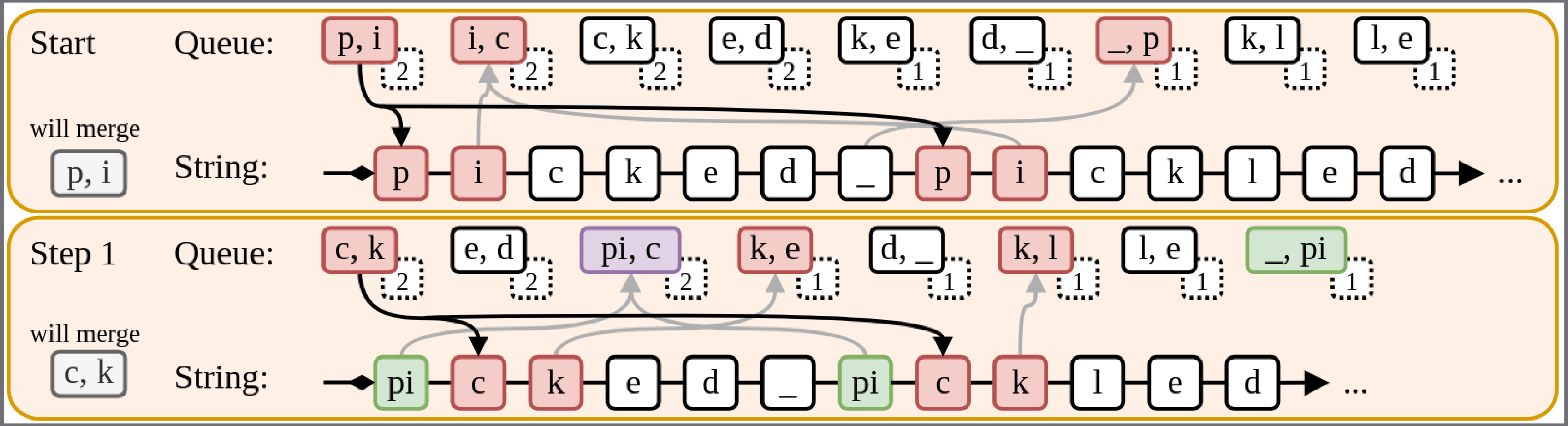
(2) submodular: $\kappa(\nu \mid \mu) \geq \kappa(\nu \mid \mu\omega) \Leftrightarrow \kappa(\mu\nu) - \kappa(\mu) \geq \kappa(\mu\omega\nu) - \kappa(\mu\omega)$

(2) hiearchical sequence-submodular: $\kappa(\nu' \mid \mu') \geq \kappa(\nu \mid \mu'\nu'\mu)$

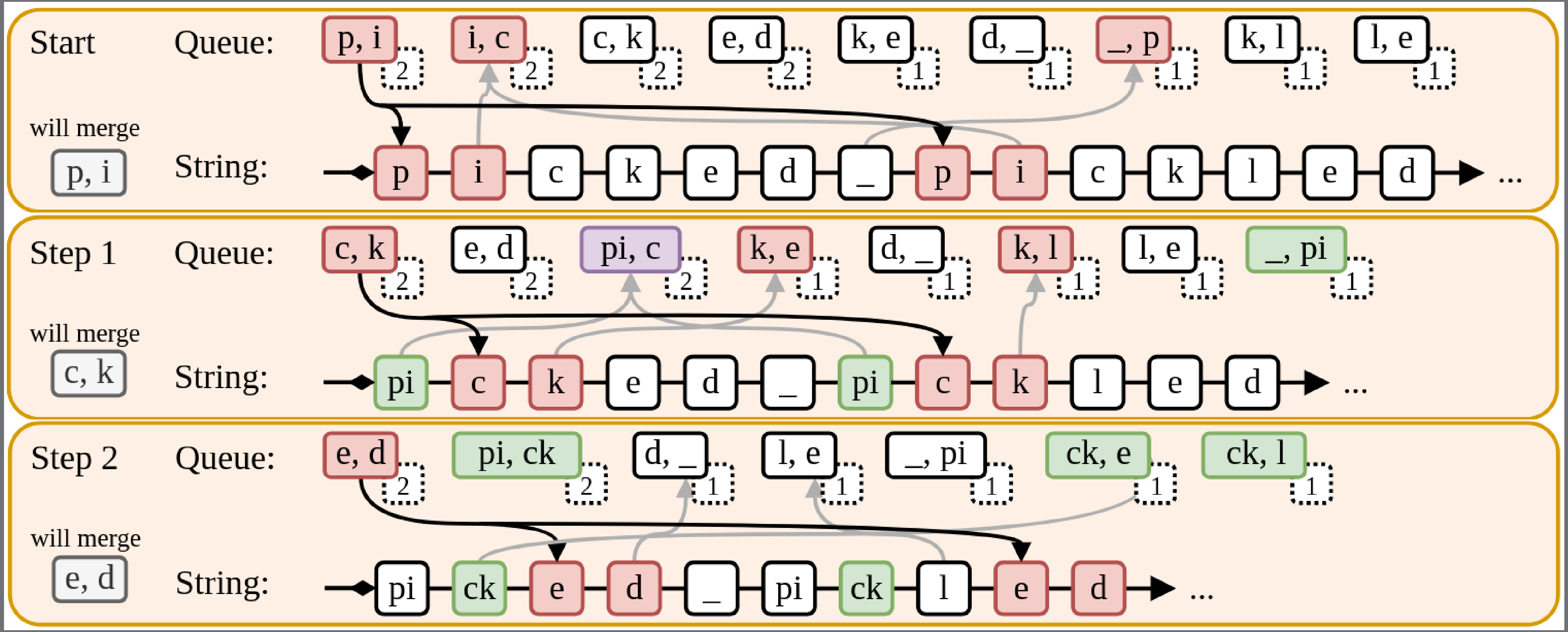
BPE can be faster



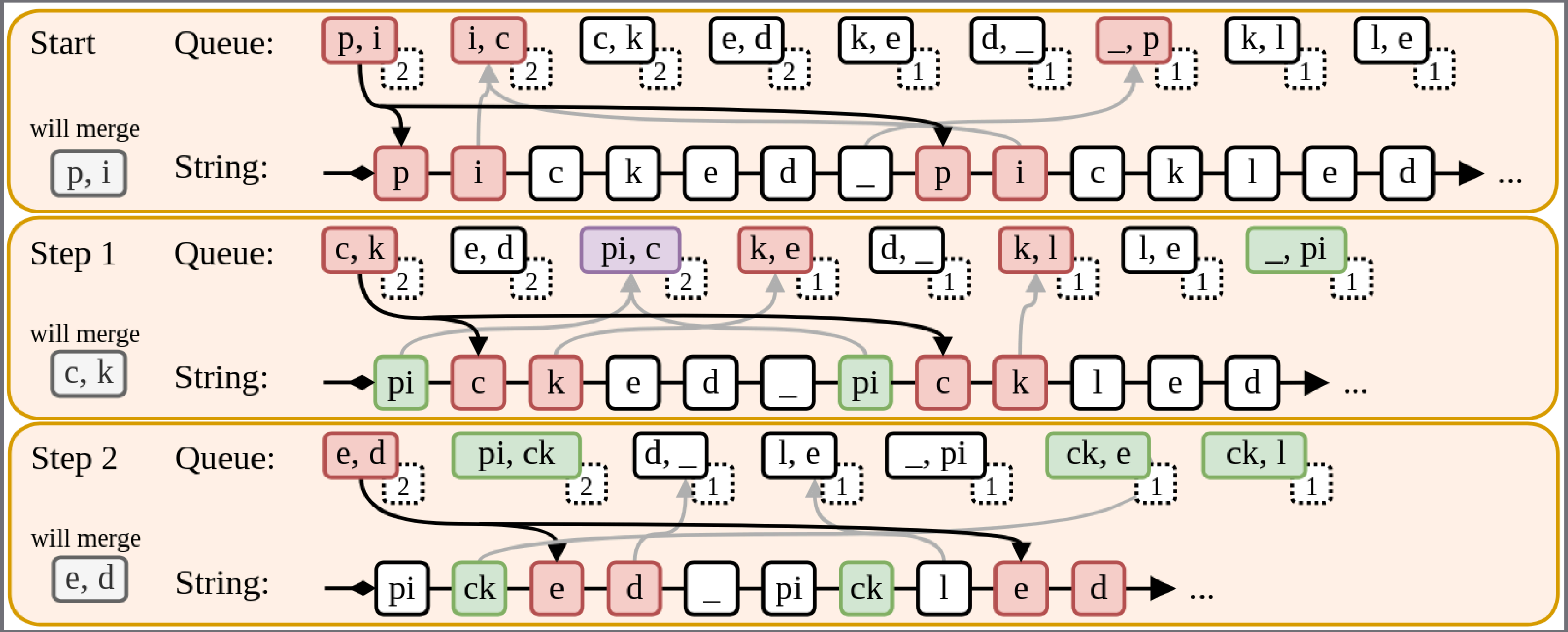
BPE can be faster



BPE can be faster



BPE can be faster



Runtime $O\left(\sum_{t=0}^{M} R_t \log M\right) = O(N \log M)$ (at most M merges)

BPE can be better: beam search

$\overline{x}$ = a b a a b b a a

$\mathrm{Freq}(x, (\mu', \mu''))$ = $\langle$ a b → 2, b a → 2, a a → 2, b b → 1 $\rangle$

BPE can be better: beam search

$\overline{x}$ = a b a a b b a a

$\mathrm{Freq}(x, (\mu', \mu''))$ = ⟨ a b → 2, b a → 2, a a → 2, b b → 1 ⟩

- Step 1:
  - Beam 1: ⟨ a b → ab ⟩ ⟨ ab a → 1, a ab → 1, ab b → 1, b a → 1, a a → 1, ⟩
  - Beam 2: ⟨ b a → b a ⟩ ⟨ ba a → 2, a ba → 1, b ba → 1, a b → 1, ⟩

BPE can be better: beam search

$\overline{x}$ = a b a a b b a a

$\mathrm{Freq}(x, (\mu', \mu''))$ = ⟨ a b → 2, b a → 2, a a → 2, b b → 1 ⟩

- Step 1:
  - Beam 1: ⟨ a b → ab ⟩ ⟨ ab a → 1, a ab → 1, ab b → 1, b a → 1, a a → 1, ⟩
  - Beam 2: ⟨ b a → b a ⟩ ⟨ ba a → 2, a ba → 1, b ba → 1, a b → 1, ⟩
- Step 1:
  - Beam 1: ⟨ a b → ab, ab a → aba ⟩ aba ab b a a
  - Beam 2: ⟨ b a → b a, ba a → baa ⟩ a baa b baa

BPE can be better: optimal

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}^*$ (optimally compressed text)

$\overline{\mu}^*$ (compression merges)

BPE can be better: optimal

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}^*$ (optimally compressed text)

$\overline{\mu}^*$ (compression merges)

$q \leftarrow \mathrm{Stack}()$
$q.\mathrm{push}(\langle\langle\rangle, \overline{x}\rangle)$
$\overline{\mu}^*, \overline{x}^* \leftarrow \langle\rangle, \overline{x}$
while q not empty **do**
    $\overline{\mu}, \overline{x} \leftarrow q.\mathrm{pop}()$
    if $|\overline{\mu}| = M$ **then continue**
    for $\mu \in \mathrm{Pairs}(\overline{x})$ **do**
        $\overline{x}' \leftarrow \mathrm{Apply}(\overline{x}, \mu)$
        $\overline{\mu}' \leftarrow \overline{\mu} \circ \mu$
        if $|\overline{x}'| < |\overline{x}^*|$ **then** $\overline{\mu}^*, \overline{x}^* \leftarrow \overline{\mu}', \overline{x}'$
        $q.\mathrm{push}(\overline{\mu}', \overline{x}')$
    end for
end while

BPE can be better: optimal

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}^*$ (optimally compressed text)

$\overline{\mu}^*$ (compression merges)

$q \leftarrow \mathrm{Stack}()$
$q.\mathrm{push}(\langle\langle\rangle, \overline{x}\rangle)$
$\overline{\mu}^*, \overline{x}^* \leftarrow \langle\rangle, \overline{x}$
while q not empty **do**
    $\overline{\mu}, \overline{x} \leftarrow q.\mathrm{pop}()$
    if $|\overline{\mu}| = M$ **then continue**
    for $\mu \in \mathrm{Pairs}(\overline{x})$ **do**
        $\overline{x}' \leftarrow \mathrm{Apply}(\overline{x}, \mu)$
        $\overline{\mu}' \leftarrow \overline{\mu} \circ \mu$
        if $|\overline{x}'| < |\overline{x}^*|$ **then** $\overline{\mu}^*, \overline{x}^* \leftarrow \overline{\mu}', \overline{x}'$
        $q.\mathrm{push}(\overline{\mu}', \overline{x}')$
    end for
end while

Runtime: $O(N^M)$

BPE can be better: optimal

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}^*$ (optimally compressed text)

$\overline{\mu}^*$ (compression merges)

BPE can be better: optimal

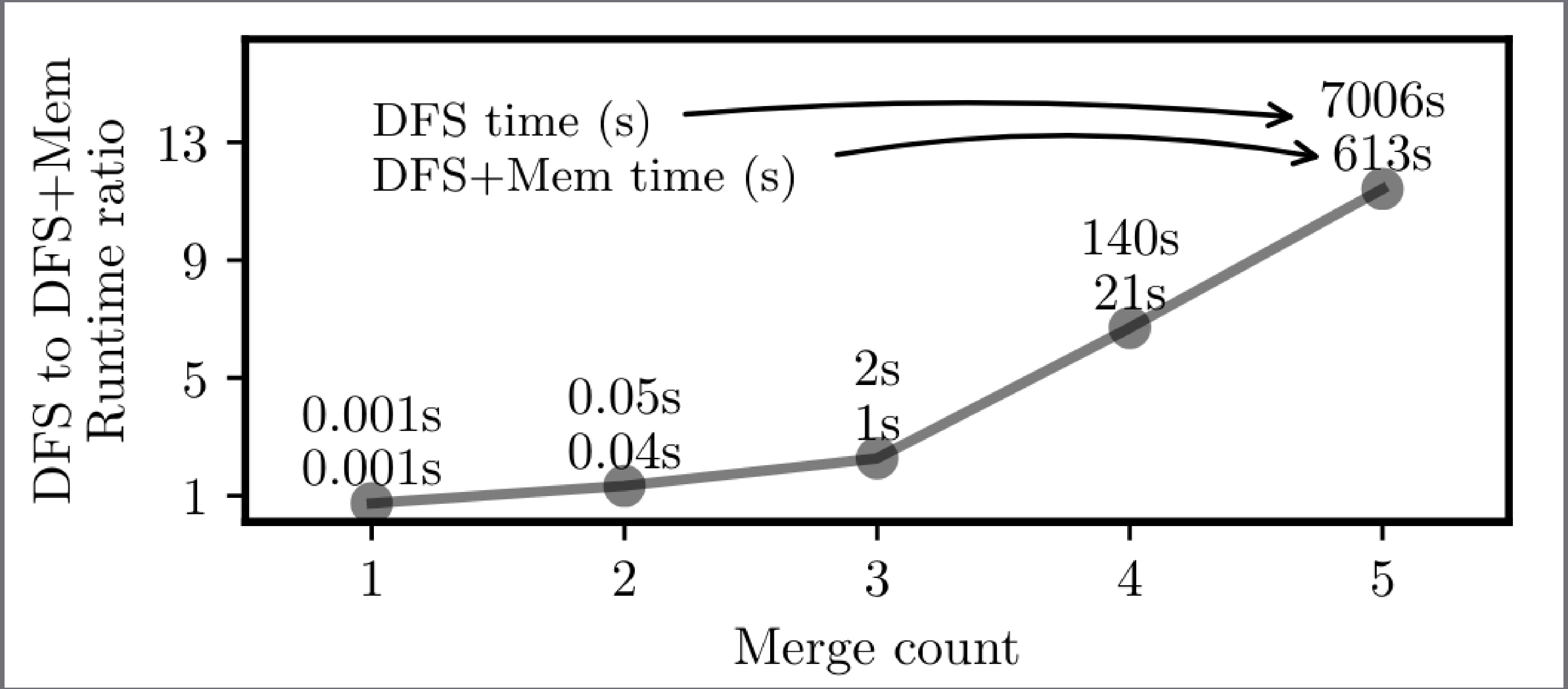
Input: $\overline{x}$ (text to compress)

Output: $\overline{x}^*$ (optimally compressed text)

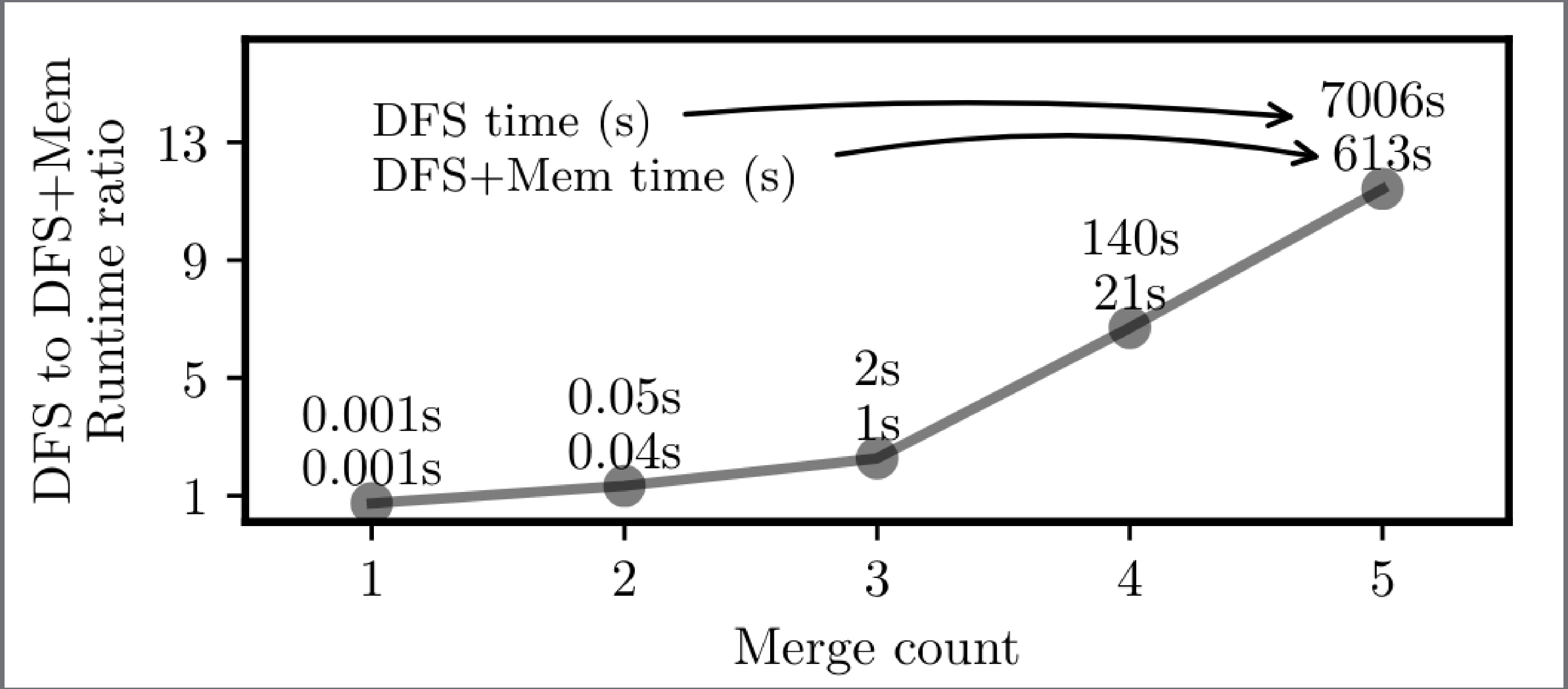
$\overline{\mu}^*$ (compression merges)

$q \leftarrow \mathrm{Stack}()$
$q.\mathrm{push}(\langle\langle\rangle, \overline{x}\rangle)$
$\overline{\mu}^*, \overline{x}^* \leftarrow \langle\rangle, \overline{x}$
while q not empty **do**
    $\overline{\mu}, \overline{x} \leftarrow q.\mathrm{pop}()$
    if $|\overline{\mu}| = M$ **then continue**
    for $\mu \in \mathrm{Pairs}(\overline{x})$ **do**
        if $|\overline{\mu}| \neq 0 \land \neg\mu \gtrdot \overline{\mu}_{-1}$ **then continue**
        $\overline{x}' \leftarrow \mathrm{Apply}(\overline{x}, \mu)$
        $\overline{\mu}' \leftarrow \overline{\mu} \circ \mu$
        if $|\overline{x}'| < |\overline{x}^*|$ **then** $\overline{\mu}^*, \overline{x}^* \leftarrow \overline{\mu}', \overline{x}'$
        $q.\mathrm{push}(\overline{\mu}', \overline{x}')$
    end for
end while

BPE on natural languages



BPE on natural languages



- Greedy BPE likely close to optimal anyway (proof/evidence missing)

Choosing the right Tokenization

Choosing the right Tokenization

before expensive model training

Choosing the right Tokenization

before expensive model training

```
th ey _ pi ck ed_    pi ck l ed _     pi ck l es
```

or

```
t h e y _ pick e d_    pickl e d _     pickl e s
```

The best tokenization..

“he loved pickled pickles”

..compresses text the most; shortest sequences

The best tokenization..

"he loved pickled pickles"

..compresses text the most; shortest sequences

each word is in the vocabulary; no segmentation

[`he, _, loved, _, pickled, _, pickles`]

The best tokenization..

“he loved pickled pickles”

..compresses text the most; shortest sequences

each word is in the vocabulary; no segmentation

[`he, _, loved, _, pickled, _, pickles`]

..compresses text the most; no redundancy;
no high-/low-frequency tokens

The best tokenization..

"he loved pickled pickles"

..compresses text the most; shortest sequences

each word is in the vocabulary; no segmentation

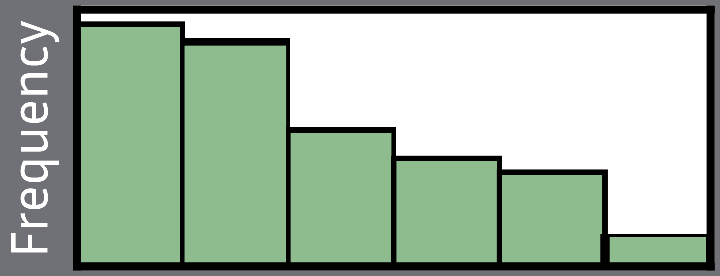
[`he, _, loved, _, pickled, _, pickles`]

..compresses text the most; no redundancy;
no high-/low-frequency tokens

entropy: $\sum_{x \in V} p(x) \log p(x)$

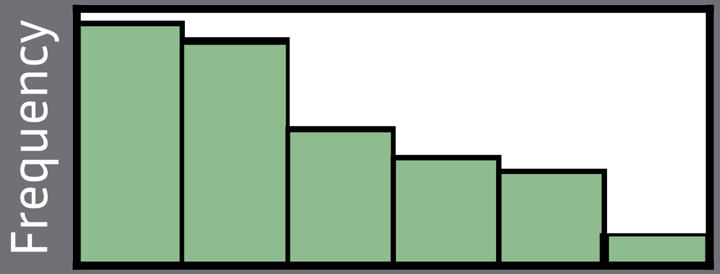
[`he, _, lov, ed, _, pickl, ed, _, pickl, es`]

Why entropy is a good tokenization metric



p i c k l e d _ p i c k l e s

Why entropy is a good tokenization metric

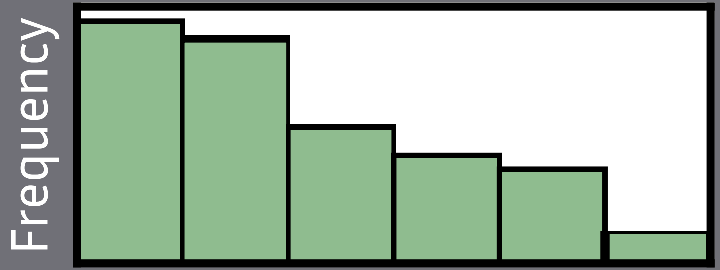


p i c k l e d _ p i c k l e s

high- & low-frequency units

no discernable meaning of each token

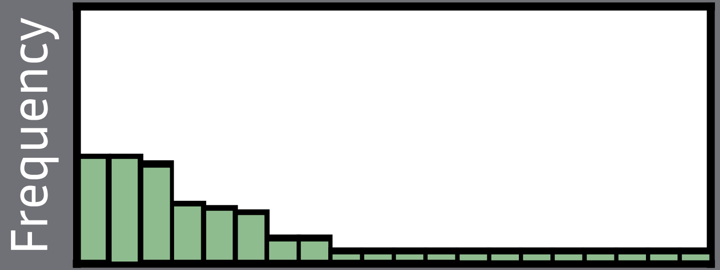
Why entropy is a good tokenization metric



`p i c k l e d _ p i c k l e s`

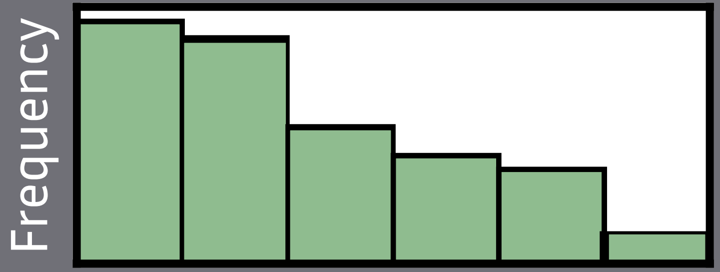
high- & low-frequency units

no discernable meaning of each token



`pickled _ pickles`

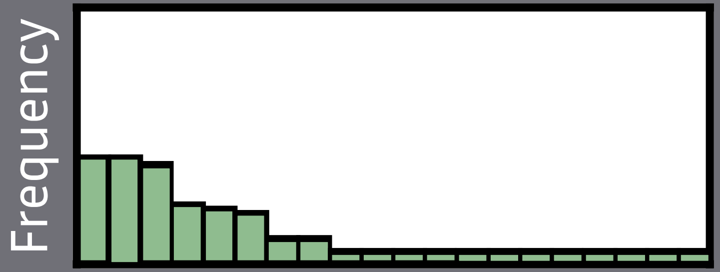
Why entropy is a good tokenization metric



p i c k l e d _ p i c k l e s

high- & low-frequency units

no discernable meaning of each token

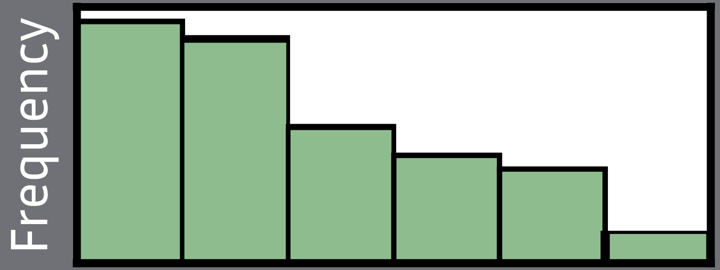


pickled _ pickles

low-frequency units

no overlap between words

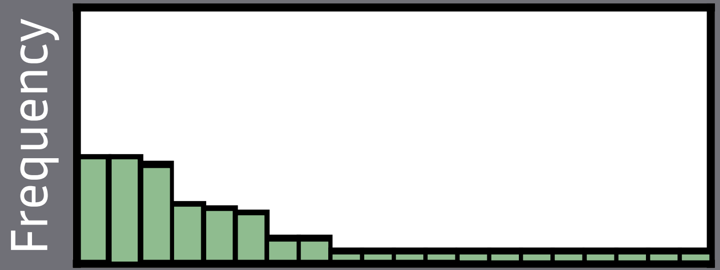
Why entropy is a good tokenization metric



p i c k l e d _ p i c k l e s

high- & low-frequency units

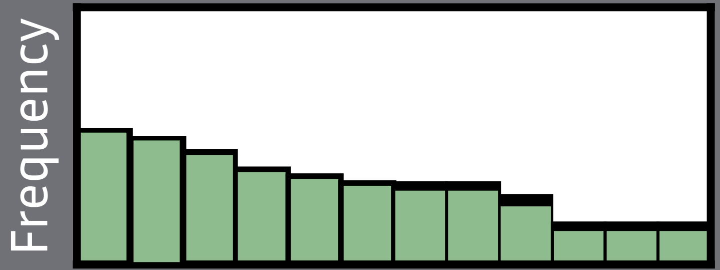
no discernable meaning of each token



pickled _ pickles

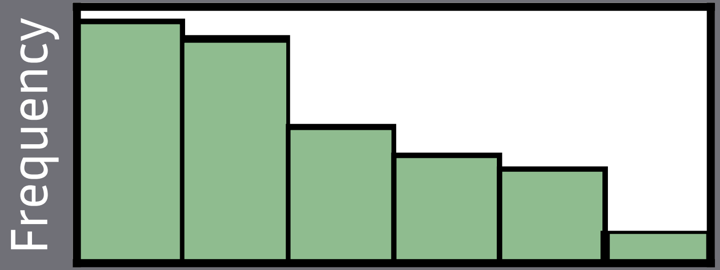
low-frequency units

no overlap between words



pi ck l ed _ pi ck l es

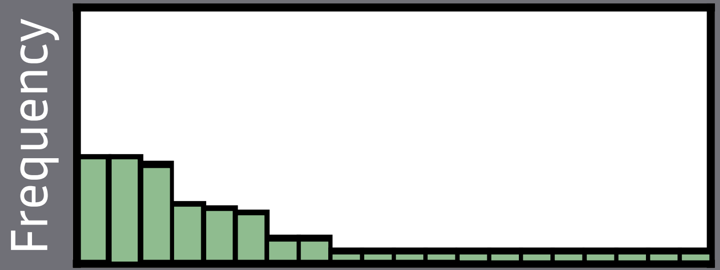
Why entropy is a good tokenization metric



p i c k l e d _ p i c k l e s

high- & low-frequency units

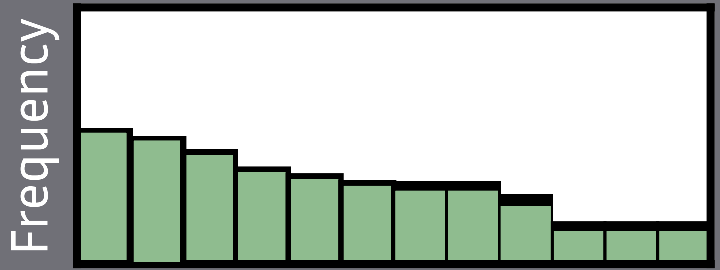
no discernable meaning of each token



pickled _ pickles

low-frequency units

no overlap between words

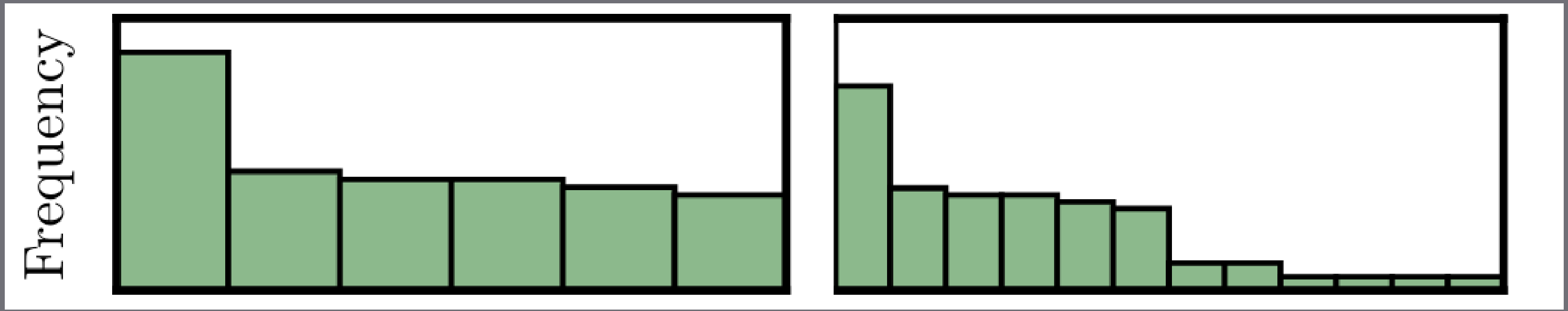


pi ck l ed _ pi ck l es

balanced, meaningful units

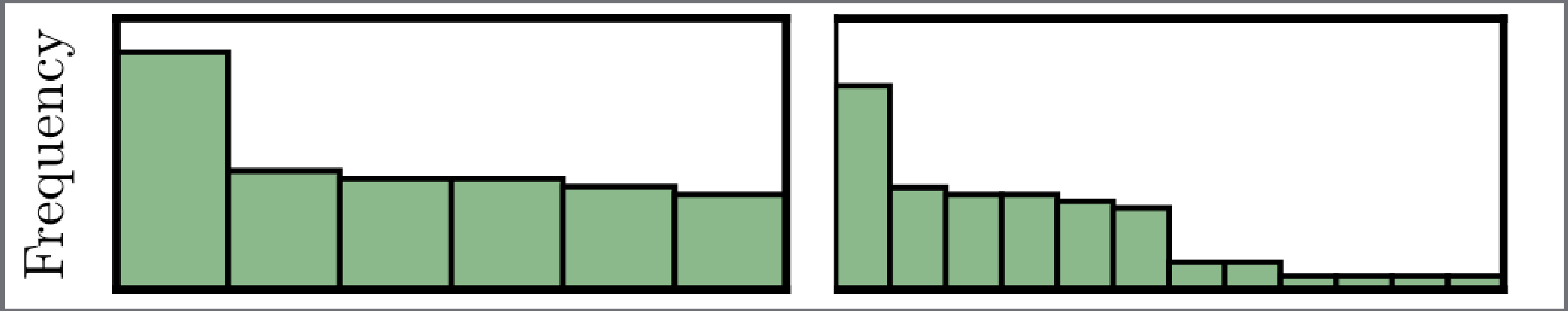
Entropies with varying vocabularies

Entropy bounds depend on support set size $H \in [0, \log|V|]$



Entropies with varying vocabularies

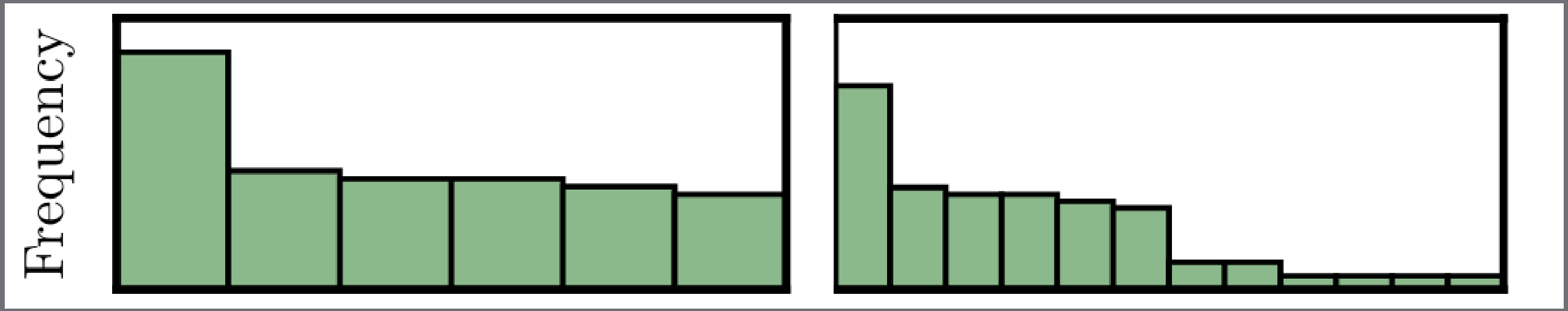
Entropy bounds depend on support set size $H \in [0, \log|V|]$



$|V_{\text{left}}| = 6 \quad |V_{\text{right}}| = 12$

Entropies with varying vocabularies

Entropy bounds depend on support set size $H \in [0, \log|V|]$

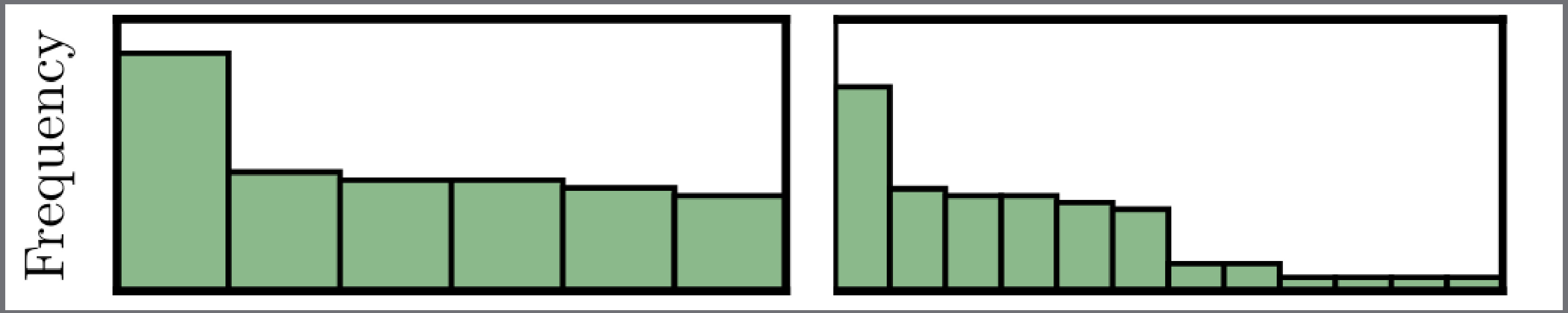


$|V_{\text{left}}| = 6 \quad |V_{\text{right}}| = 12$

$H_{\text{left}} = 2.50 \quad H_{\text{right}} = 3.08$

Entropies with varying vocabularies

Entropy bounds depend on support set size $H \in [0, \log|V|]$

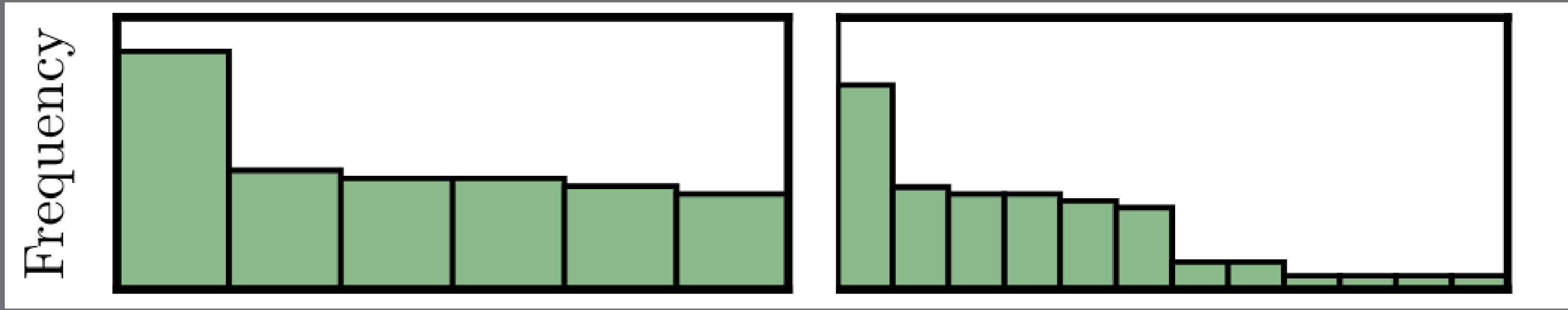


$|V_{\mathrm{left}}| = 6 \quad |V_{\mathrm{right}}| = 12$ $H_{\mathrm{left}} = 2.50 \quad H_{\mathrm{right}} = 3.08$

proportion to maximal possible entropy given $|V|$

Entropies with varying vocabularies

Entropy bounds depend on support set size $H \in [0, \log|V|]$



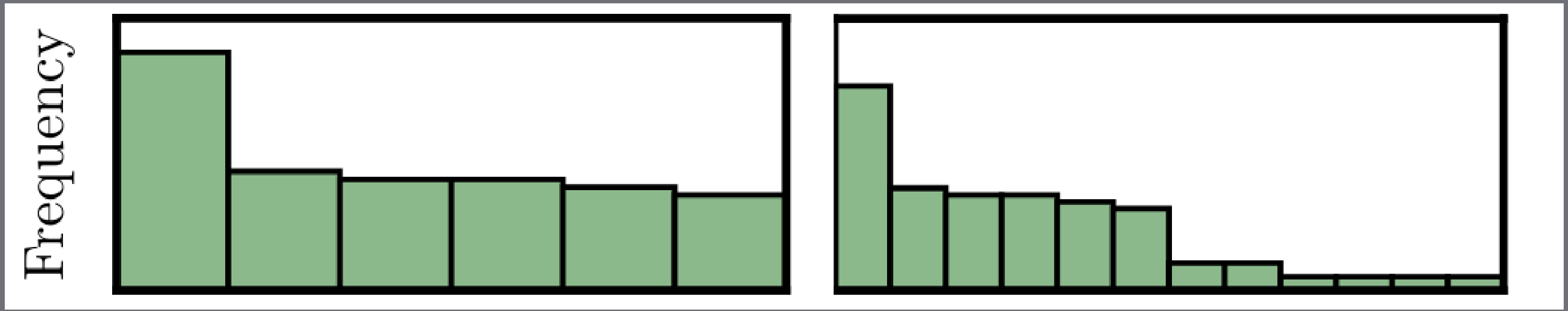
$|V_{\text{left}}| = 6 \quad |V_{\text{right}}| = 12$ $H_{\text{left}} = 2.50 \quad H_{\text{right}} = 3.08$

proportion to maximal possible entropy given $|V|$

efficiency: $\frac{\text{optimal encoding of } p}{\text{uninformed encoding of } p} = \frac{H_p}{H_U} = \frac{H_p}{\log|V|}$

Entropies with varying vocabularies

Entropy bounds depend on support set size $H \in [0, \log|V|]$



$|V_{\text{left}}| = 6 \quad |V_{\text{right}}| = 12$ $H_{\text{left}} = 2.50 \quad H_{\text{right}} = 3.08$

proportion to maximal possible entropy given $|V|$

efficiency: $\frac{\text{optimal encoding of } p}{\text{uninformed encoding of } p} = \frac{H_p}{H_U} = \frac{H_p}{\log|V|}$

$\text{Eff}_{\text{left}} = 97\% \quad \text{Eff}_{\text{right}} = 86\%$

Entropies with varying penalization

Shannon Entropy: $-\sum p \log p$ $\approx$ expected optimal encoding length

low frequency p gets long codes

Entropies with varying penalization

Shannon Entropy: $-\sum p \log p$

$\approx$ expected optimal encoding length

low frequency p gets long codes

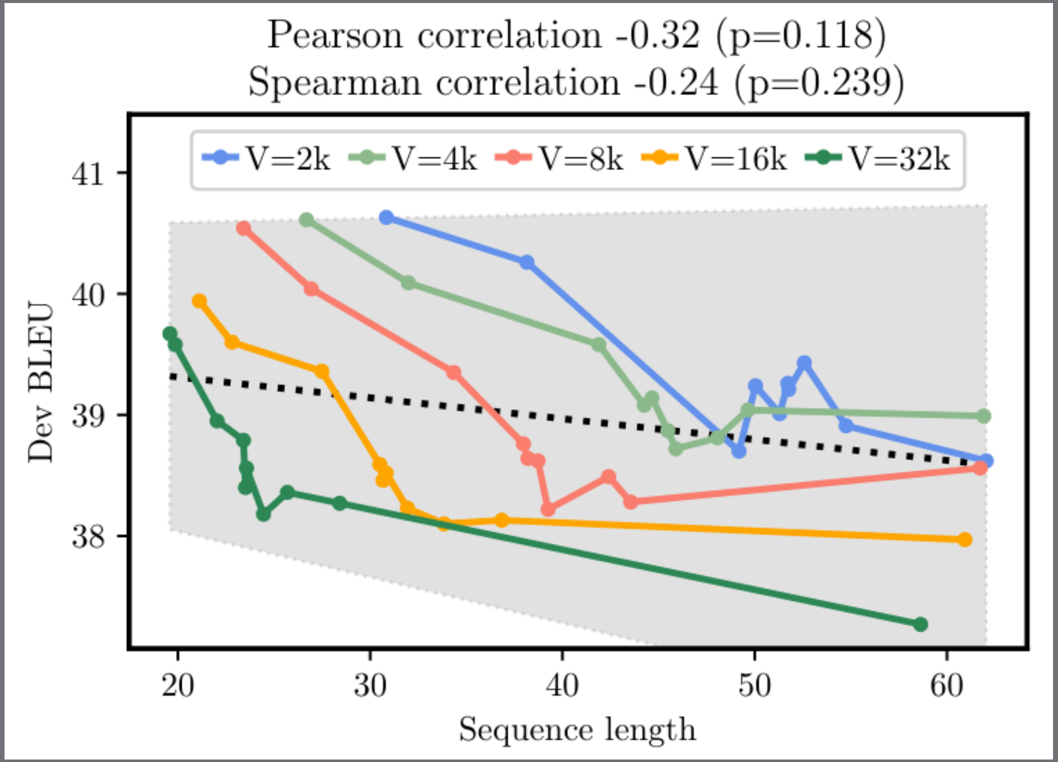
Rényi Entropy: $\frac{1}{1-\alpha} \log(\sum p^\alpha)$

$\approx$ expected optimal encoding length

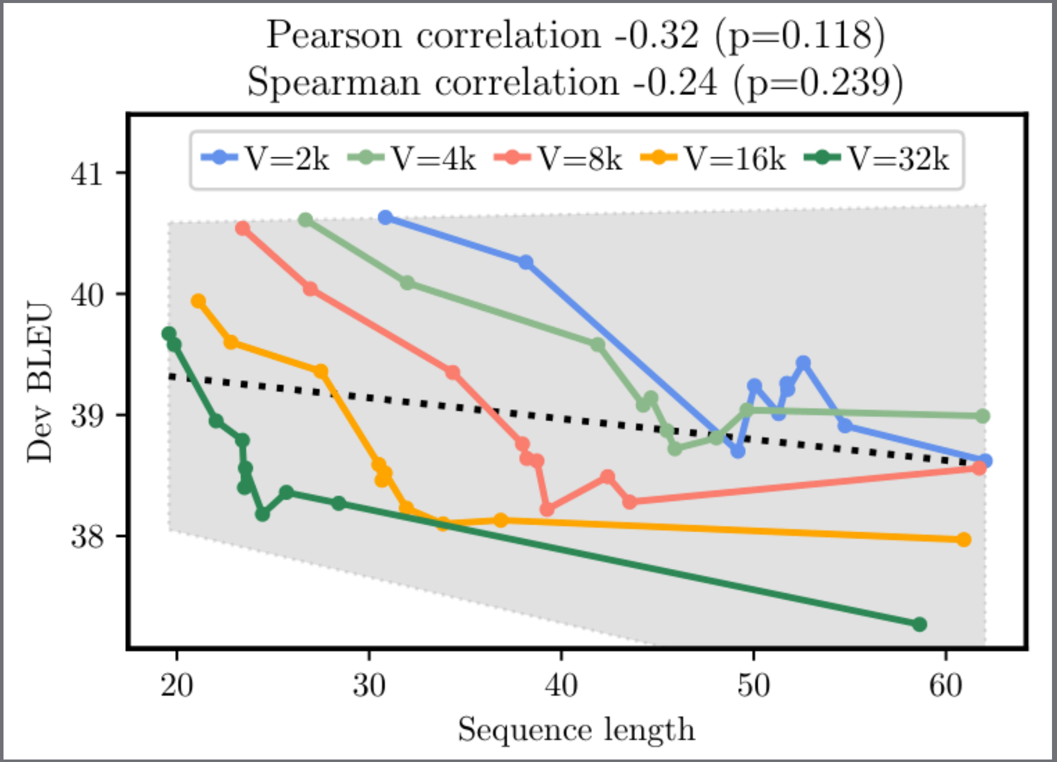
low frequency p gets long codes

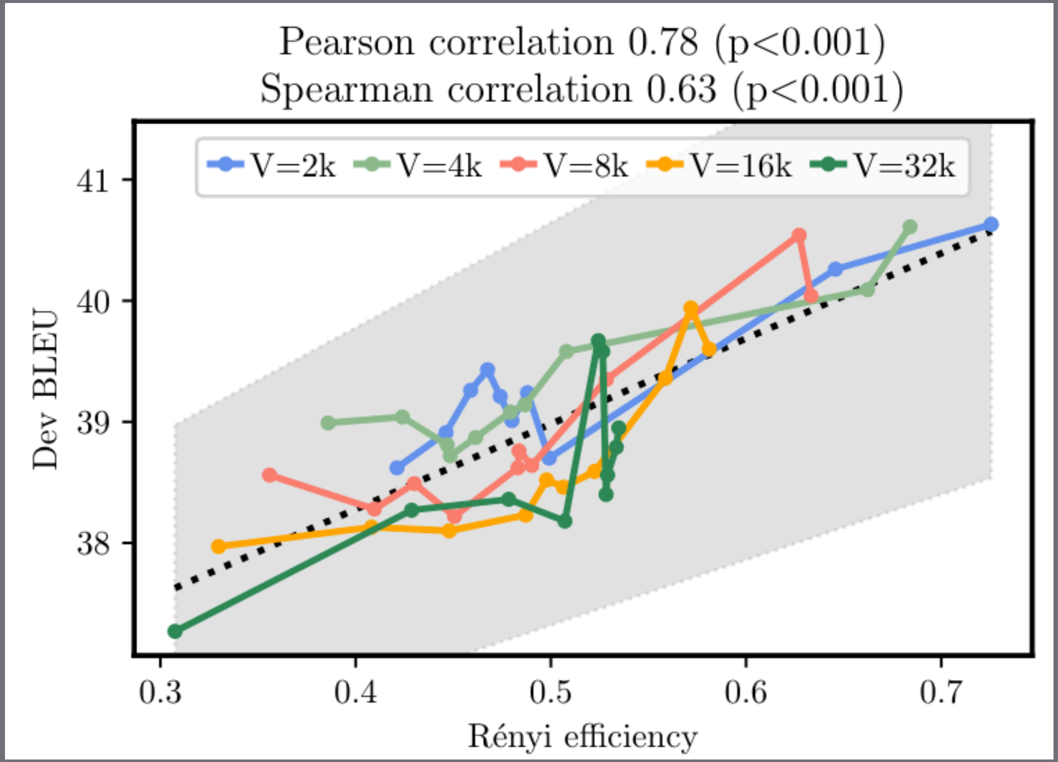
too long/short codes (depending on α) are penalized

Choosing good tokenization



Choosing good tokenization





Choosing good tokenization

Predictor	Pearson	Spearman	ρ^2
Sequence len.	-0.32 (=0.118)	-0.24 (=0.239)	10%
Percentile freq.	0.76 (<0.001)	0.63 (<0.001)	58%
Entropy	0.22 (=0.281)	0.12 (=0.578)	5%
Entropy eff.	0.56 (=0.004)	0.38 (=0.006)	31%
Rényi entropy	0.49 (=0.001)	0.38 (=0.006)	24%
Rényi eff.	0.78 (<0.001)	0.66 (<0.001)	61%

tokenization-scorer

CLI

```
$ head -n 1 data1.txt
pick @@ed pick @@l @@ed pick @@les
$ head -n 1 data2.txt
pick @@e @@d pick @@l @@e @@d pick @@l
@@e @@s

$ pip install tokenization-scorer

$ tokenization-scorer -i data1.txt
0.8031528501359657
$ tokenization-scorer -i data2.txt
0.9105681923824472
```

tokenization-scorer

CLI

```
$ head -n 1 data1.txt
pick @@ed pick @@l @@ed pick @@les
$ head -n 1 data2.txt
pick @@e @@d pick @@l @@e @@d pick @@l
@@e @@s

$ pip install tokenization-scorer

$ tokenization-scorer -i data1.txt
0.8031528501359657
$ tokenization-scorer -i data2.txt
0.9105681923824472
```

Python

```
import tokenization_scorer
data1 = "pick @@ed pick @@l @@ed pick @@les"
tokenization_scorer.score(
  text1, metric="renyi", power=2.5
)
> 0.8031528501359657

data2 = "pick @@e @@d pick @@l @@e @@d pick @@l @@e @@s"
tokenization_scorer.score(
  text2, metric="renyi", power=2.5
)
> 0.9105681923824472
```

tokenization-scorer

CLI

```
$ head -n 1 data1.txt
pick @@ed pick @@l @@ed pick @@les
$ head -n 1 data2.txt
pick @@e @@d pick @@l @@e @@d pick @@l
@@e @@s

$ pip install tokenization-scorer

$ tokenization-scorer -i data1.txt
0.8031528501359657
$ tokenization-scorer -i data2.txt
0.9105681923824472
```

Python

```
import tokenization_scorer
data1 = "pick @@ed pick @@l @@ed pick @@les"
tokenization_scorer.score(
  text1, metric="renyi", power=2.5
)
> 0.8031528501359657

data2 = "pick @@e @@d pick @@l @@e @@d pick @@l @@e @@s"
tokenization_scorer.score(
  text2, metric="renyi", power=2.5
)
> 0.9105681923824472
```

Paper: Tokenization and the Noiseless Channel

tokenization-scorer

demo

A few questions about Tokenization and the Noiseless Channel

Marco <cognetta.marco@gmail.com> 25 Jul 2023, 04:17

to vzouhar

Hi Vilém,

A few questions about Tokenization and the Noiseless Channel

Marco <cognetta.marco@gmail.com> 25 Jul 2023, 04:17

to vzouhar

Hi Vilém,

I think your paper is incorrect

A few questions about Tokenization and the Noiseless Channel

Marco <cognetta.marco@gmail.com> 25 Jul 2023, 04:17

to vzouhar

Hi Vilém,

I think your paper is incorrect

and I can prove it

A few questions about Tokenization and the Noiseless Channel

Marco <cognetta.marco@gmail.com> 25 Jul 2023, 04:17

to vzouhar

Hi Vilém,

I think your paper is incorrect

and I can prove it

Best, Marco

(untrue dramatic reenactment)

Counterexample?

Hypothesis: higher efficiency → higher performance

Counterexample?

Hypothesis:	higher efficiency → higher performance
Counterexample:	increase efficiency *but* lower performance

Counterexample?

Hypothesis: higher efficiency → higher performance

Counterexample: increase efficiency *but* lower performance

```
she  pick @@ed pick @@l @@ed pick @@les
he   pick @@ed pick @@l @@ed pick @@les
they pick @@ed pick @@l @@ed pick @@les
```

→

duplicate `pick` into `pick{1}` and `pick{2}`
→ (swap randomly)

```
she  pick{1} @@ed pick{1} @@l @@ed pick{1} @@les
he   pick{2} @@ed pick{1} @@l @@ed pick{2} @@les
they pick{2} @@ed pick{2} @@l @@ed pick{1} @@les
```

Counterexample?

- `the`: 5%
- `a`: 4%
- `-es`: 1%
- `-ing`: 1%
- …

Counterexample?

- `the`: 5%
- `a`: 4%
- `-es`: 1%
- `-ing`: 1%
- …

→

- `the{1}`: 1%
- `the{2}`: 1%
- `the{3}`: 1%
- `the{4}`: 1%
- `the{5}`: 1%
- `a`: 4%
- `-es`: 1%
- `-ing`: 1%
- …

Counterexample?

- `the`: 5%
- `a`: 4%
- `-es`: 1%
- `-ing`: 1%
- …

→

- `the{1}`: 1%
- `the{2}`: 1%
- `the{3}`: 1%
- `the{4}`: 1%
- `the{5}`: 1%
- `a`: 4%
- `-es`: 1%
- `-ing`: 1%
- …

→ increase entropy (vocabulary only in log)

→ $\frac{H(p)}{\log V}$ goes up

→ just noise, so lower performance

Counterexample?

	Original BPE	Duplicated BPE
Efficiency	40%	50%

Counterexample?

	Original BPE	Duplicated BPE
Efficiency	40%	50%
BLEU	33.59	32.49

Counterexample?

	Original BPE	Duplicated BPE
Efficiency	40%	50%
BLEU	33.59	32.49

All is lost?

Counterexample?

	Original BPE	Duplicated BPE
Efficiency	40%	50%
BLEU	33.59	32.49

All is lost?

- non-natural adversarial examples

Counterexample?

	Original BPE	Duplicated BPE
Efficiency	40%	50%
BLEU	33.59	32.49

All is lost?

- non-natural adversarial examples
- invitation to find a better metric

Paper: Two Counterexamples Tokenization and the Noiseless Channel (Cognetta et al.)

Stochastic Tokenization

- So far $\mathrm{tokenize}(\texttt{pickles}) = \langle\texttt{pick les}\rangle$ (deterministic)

Stochastic Tokenization

- So far $\mathrm{tokenize}(\texttt{pickles}) = \langle \texttt{pick les} \rangle$ (deterministic)
- Vocabulary $V = \Sigma \cup \{\texttt{pi}, \texttt{ck}, \texttt{pick}, \texttt{le}\}$
- $\{\texttt{pick les}, \texttt{pi ck les}, \texttt{pick le s}, \texttt{p i c k les}, \ldots\} \subset \Sigma^*$

Stochastic Tokenization

- So far $\mathrm{tokenize}(\texttt{pickles}) = \langle \texttt{pick les} \rangle$ (deterministic)
- Vocabulary $V = \Sigma \cup \{\texttt{pi}, \texttt{ck}, \texttt{pick}, \texttt{le}\}$
- $\{\texttt{pick les}, \texttt{pi ck les}, \texttt{pick le s}, \texttt{p i c k les}, \ldots\} \subset \Sigma^*$
- $\cancel{\mathrm{tokenize}}(\cancel{\texttt{pickles, pickles, pickles, pickles, pickles}})$
  $= \langle \cancel{\texttt{pick les , pick les , pick les , pick les , pick les}} \rangle$
- $\mathrm{tokenize}'(\texttt{pickles, pickles, pickles, pickles, pickles})$
  $= \langle \texttt{pi ck les , pick l e s , pick les , p i ck l e s , pi ck les} \rangle$

Stochastic Tokenization

- So far $\text{tokenize}(\texttt{pickles}) = \langle\texttt{pick}\ \texttt{les}\rangle$ (deterministic)
- Vocabulary $V = \Sigma \cup \{\texttt{pi}, \texttt{ck}, \texttt{pick}, \texttt{le}\}$
- $\{\texttt{pick les}, \texttt{pi ck les}, \texttt{pick le s}, \texttt{p i c k les}, \ldots\} \subset \Sigma^*$
- $\cancel{\text{tokenize}}(\cancel{\texttt{pickles, pickles, pickles, pickles, pickles}})$
  $= \langle\cancel{\texttt{pick les , pick les , pick les , pick les , pick les}}\rangle$
- $\text{tokenize}'(\texttt{pickles, pickles, pickles, pickles, pickles})$
  $= \langle\texttt{pi ck les , pick l e s , pick les , p i ck l e s , pi ck les}\rangle$
- Deterministic: $\text{tokenize} : \Sigma^* \to V^*$
- Stochastic: $\text{tokenize} : \Sigma^* \to \underbrace{[X \to V^*]}_{\text{distribution}}$
- Increases robustness

Stochastic Tokenization

- $\text{tokenize}(\texttt{pickles})$
  $= \{80\% : \texttt{pick les}, 10\% : \texttt{pi ck les}, 5\% : \texttt{pick le s}, \ldots\}$

Stochastic Tokenization

- $\text{tokenize}(\texttt{pickles})$

  $= \{80\% : \texttt{pick les}, 10\% : \texttt{pi ck les}, 5\% : \texttt{pick le s}, \ldots\}$
- BPE Dropout (modified from Provilkov et al., 2020)

Input: $\overline{x}$ (word), merges $\overline{\mu}$

Output: $\overline{x}$ (tokenized word)

1. $\overline{\mu}' \leftarrow \langle (\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \text{Rand}() < p \rangle$
2. **while** $\overline{\mu}' \neq \emptyset$ **do**
3. $\quad (a, b) \leftarrow \overline{\mu}'_1$
4. $\quad \overline{x} \leftarrow \text{ApplyMerge}(\text{a b} \rightarrow \text{ab}, \overline{x})$
5. $\quad \overline{\mu}' \leftarrow \langle (\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \text{Rand}() < p \rangle$
6. **end while**
7. **return** $\overline{x}$

Stochastic Tokenization: BPE-Dropout

Input: $\overline{x}$ (word), merges $\overline{\mu}$ **Output**: $\overline{x}$ (tokenized word)

1. $\overline{\mu}' \leftarrow \langle (\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \;\wedge \mathrm{Rand}() < p \rangle$
2. **while** $\overline{\mu}' \neq \emptyset$ **do**
3. $\quad (a, b) \leftarrow \overline{\mu}'_1$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mathrm{a\ b} \rightarrow \mathrm{ab}, \overline{x})$
5. $\quad \overline{\mu}' \leftarrow \langle (\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \;\wedge \mathrm{Rand}() < p \rangle$
6. **end while**
7. **return** $\overline{x}$

Stochastic Tokenization: BPE-Dropout

Input: $\overline{x}$ (word), merges $\overline{\mu}$ **Output:** $\overline{x}$ (tokenized word)

1. $\overline{\mu}' \leftarrow \langle(\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \mathrm{Rand}() < p \rangle$
2. **while** $\overline{\mu}' \neq \emptyset$ **do**
3. $\quad (a, b) \leftarrow \overline{\mu}'_1$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mathrm{a\ b} \rightarrow \mathrm{ab}, \overline{x})$
5. $\quad \overline{\mu}' \leftarrow \langle(\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \mathrm{Rand}() < p \rangle$
6. **end while**
7. **return** $\overline{x}$

~~$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pi ck` → `pick`, `pick l` → `pickl` ⟩~~

$\overline{\mu}$ = ⟨ `p i` → `pi`, `c k` → `ck`, `pick l` → `pickl` ⟩

Stochastic Tokenization: BPE-Dropout

Input: $\overline{x}$ (word), merges $\overline{\mu}$ **Output:** $\overline{x}$ (tokenized word)

1. $\overline{\mu}' \leftarrow \langle(\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \mathrm{Rand}() < p \rangle$
2. **while** $\overline{\mu}' \neq \emptyset$ **do**
3. $\quad (a, b) \leftarrow \overline{\mu}'_1$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mathrm{a\ b} \rightarrow \mathrm{ab}, \overline{x})$
5. $\quad \overline{\mu}' \leftarrow \langle(\overline{x}_i, \overline{x}_{i+1}) \mid (\overline{x}_i, \overline{x}_{i+1}) \in \overline{\mu} \wedge \mathrm{Rand}() < p \rangle$
6. **end while**
7. **return** $\overline{x}$

~~$\overline{\mu}$ = ⟨ p i → pi, c k → ck, pi ck → pick, pick l → pickl ⟩~~

$\overline{\mu}$ = ⟨ p i → pi, c k → ck, pick l → pickl ⟩

~~T(pickles) = pickl e s~~ T(pickles) = pi ck l e s

Stochastic Tokenization is not Uniform

BPE-Dropout $p = 0.1$	
to ken ization	97.77%
to ke n ization	1.89%
to k en ization	0.25%
to ken iz ation	0.04%
t oken ization	0.03%
to k en iz ation	0.01%
to ke n iz ation	0.01%
to ken i z ation	$<0.01\%$

Stochastic Tokenization can be Uniform

Input: $\overline{x}$ (word), tokenizer T

Output: $\overline{x}$ (tokenized word)

1. **if** $\mathrm{Rand}() < \mathrm{p}$ **then**
2. **return** $\mathrm{RandomTokenization}(\overline{x})$
3. **else**
4. **return** $T(\overline{x})$
5. **end if**

Stochastic Tokenization can be Uniform

Input: $\overline{x}$ (word), tokenizer T

Output: $\overline{x}$ (tokenized word)

1. **if** $\mathrm{Rand}() < \mathrm{p}$ **then**
2. **return** $\mathrm{RandomTokenization}(\overline{x})$
3. **else**
4. **return** $T(\overline{x})$
5. **end if**

How to get $\mathrm{RandomTokenization}(\overline{x})$?

Modelling All Possible Tokenizations

- Idea 1: run BPE-Dropout for each word 10^7 times and store all unique tokenizations.

Modelling All Possible Tokenizations

- Idea 1: run BPE-Dropout for each word 10^7 times and store all unique tokenizations.
  - Very slow.

Modelling All Possible Tokenizations

- Idea 1: run BPE-Dropout for each word 10^7 times and store all unique tokenizations.
  - Very slow.
- Idea 2: construct the set for each word apriori.

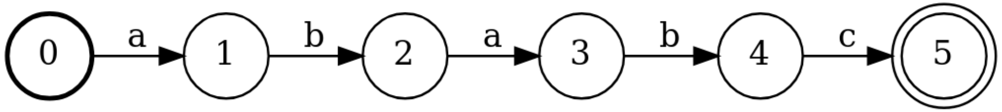
Modelling All Possible Tokenizations

- Idea 1: run BPE-Dropout for each word 10^7 times and store all unique tokenizations.
  - Very slow.
- Idea 2: construct the set for each word apriori.
  - Very big; unclear how.
  - For word of length $|\overline{x}|$ the tight upper bound is $2^{|\overline{x}|-1}$ (yes/no space).

Modelling All Possible Tokenizations

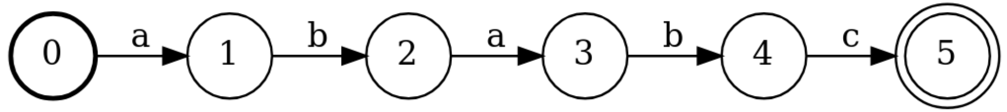
- Idea 1: run BPE-Dropout for each word 10^7 times and store all unique tokenizations.
  - Very slow.
- Idea 2: construct the set for each word apriori.
  - Very big; unclear how.
  - For word of length $|\overline{x}|$ the tight upper bound is $2^{|\overline{x}|-1}$ (yes/no space).
- Idea 3: function that can yield random tokenization.

Modelling All Possible Tokenizations

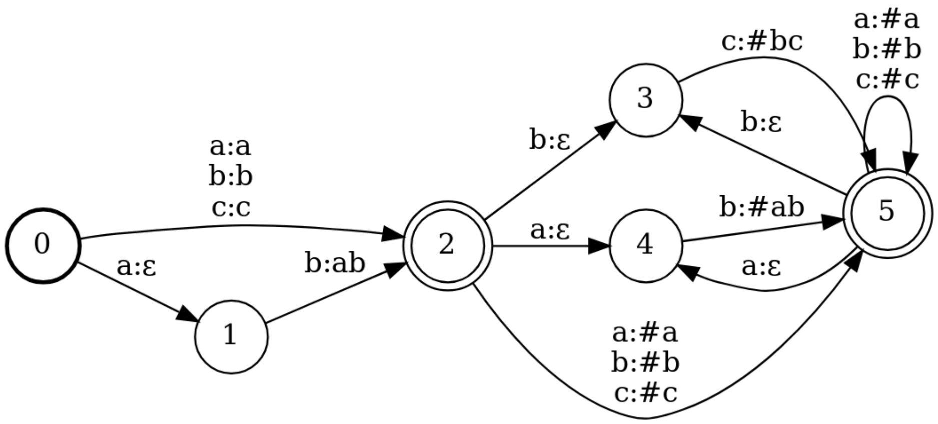


(a) An automaton $\mathcal{A}$ representing ababc.

Modelling All Possible Tokenizations

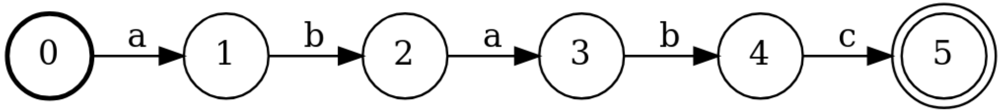


(a) An automaton $\mathcal{A}$ representing ababc.

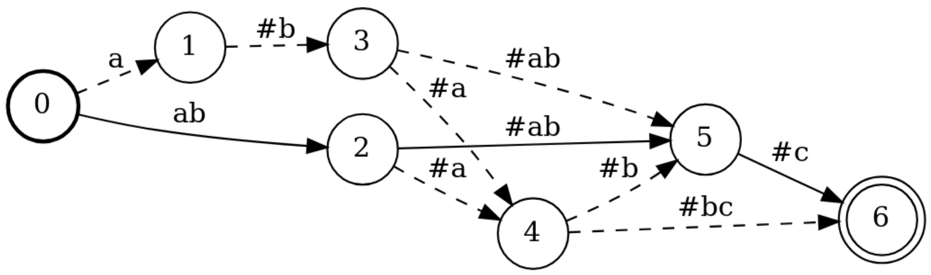


(b) A transducer $\mathcal{T}$ for the subword vocabulary {a, b, c, ab, #a, #b, #c, #ab, #bc}.

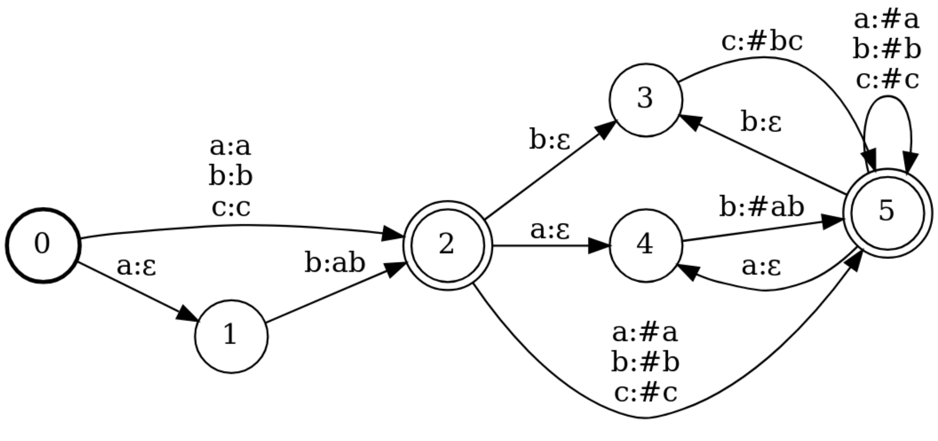
Modelling All Possible Tokenizations



(a) An automaton $\mathcal{A}$ representing ababc.

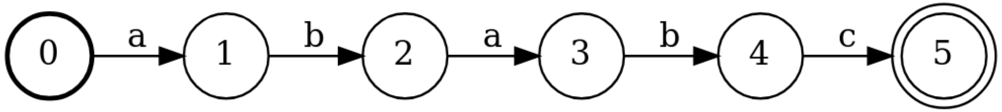


(c) A lattice, $\mathcal{A} \circ \mathcal{T}$, of all possible tokenizations of ababc.

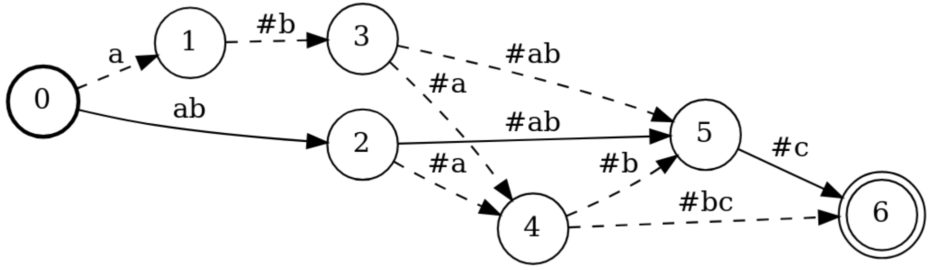


(b) A transducer $\mathcal{T}$ for the subword vocabulary {a, b, c, ab, #a, #b, #c, #ab, #bc}.

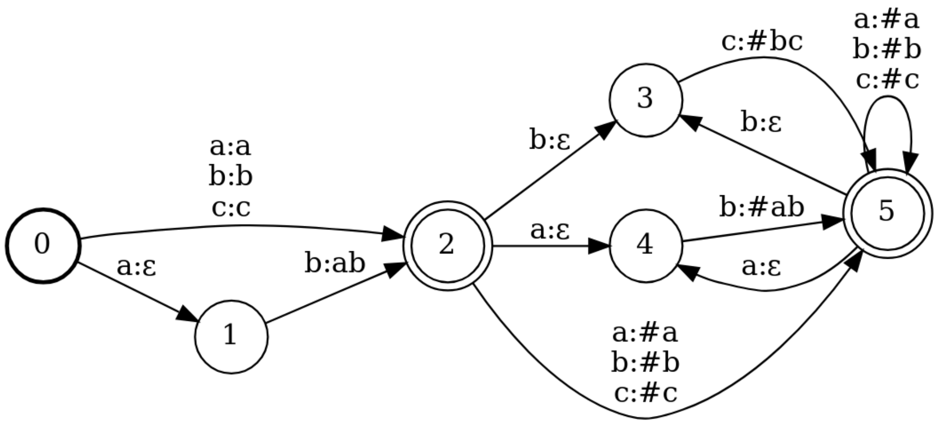
Modelling All Possible Tokenizations



(a) An automaton $\mathcal{A}$ representing `ababc`.



(c) A lattice, $\mathcal{A} \circ \mathcal{T}$, of all possible tokenizations of `ababc`.



(b) A transducer $\mathcal{T}$ for the subword vocabulary `{a, b, c, ab, #a, #b, #c, #ab, #bc}`.

- Walk through lattice is tokenization
- Can be sampled randomly in linear time

Uniform Sampling Increases Diversity

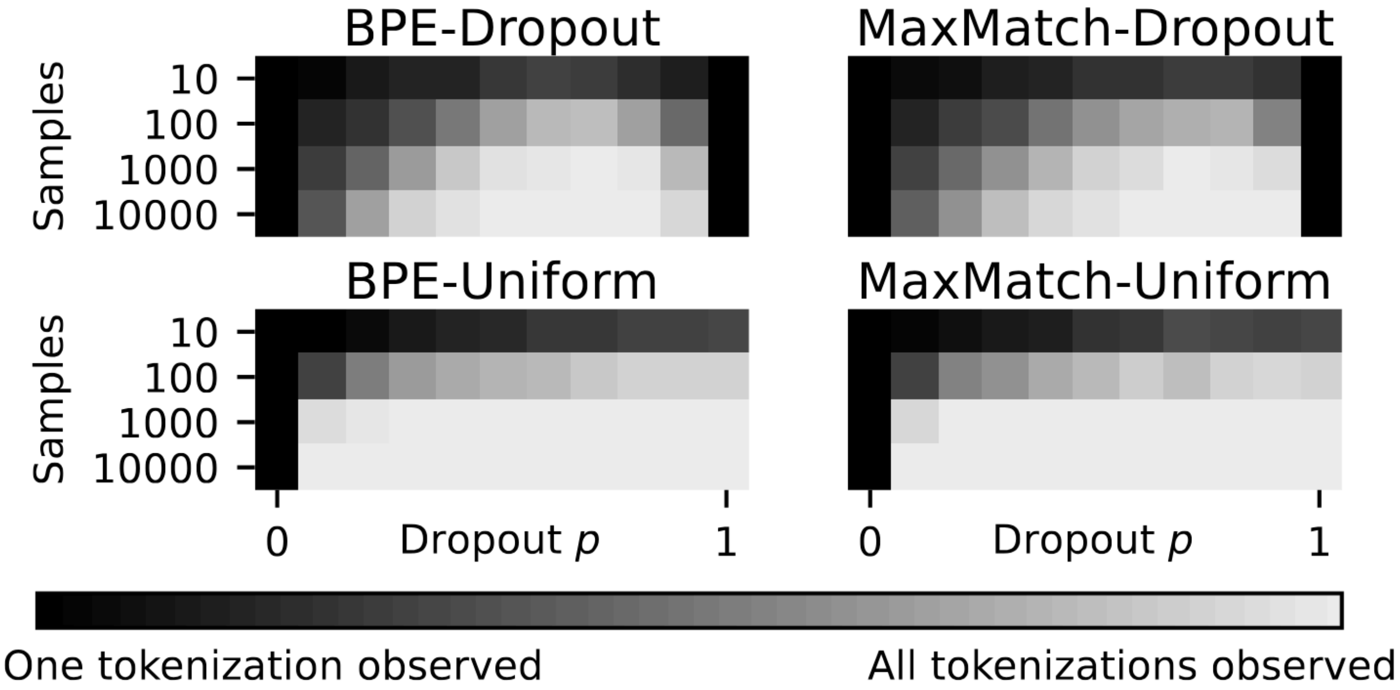


Figure 2: The number of unique, observed tokenizations of a word with N samples and dropout p.

Uniform Sampling Increases Efficiency (and Performance?)

Tokenizer	Efficiency	BLEU	chrF	COMET
BPE	0.4524	23.56	53.20	81.03
BPE + Dropout (p=0.1)	0.4614	23.98	53.70	81.90
BPE + Uniform (p=0.1)	0.4594	23.83	53.67	82.00
BPE + Uniform (p=0.25)	0.4647	**24.13**	**53.73**	**82.20**
MaxMatch	0.4476	23.52	53.23	81.17
MaxMatch + Dropout (p=0.3)	0.4578	23.95	53.70	81.98
MaxMatch + Uniform (p=0.1)	0.4528	**24.32**	**53.90**	**82.11**
MaxMatch + Uniform (p=0.25)	0.4563	24.10	53.87	82.06
Unigram (α=1)	0.4338	23.68	53.37	81.28
Unigram (α=0.3)	0.4284	24.17	53.87	82.00

Paper: Distributional Properties of Subword Regularization (Cognetta et al.)

Miscellaneous Tricks

```
$ from transformers import AutoTokenizer
$ vocab = AutoTokenizer.from_pretrained("gpt2").vocab
$ [x for x in vocab if len(x) > 50]
```

Miscellaneous Tricks

```
$ from transformers import AutoTokenizer
$ vocab = AutoTokenizer.from_pretrained("gpt2").vocab
$ [x for x in vocab if len(x) > 50]
```

yields:

```
=================================================================
................................................................
----------------------------------------------------------------
________________________________________________________________
================================================================
----------------------------------------------------------------
ÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤ
--------------------------------------------------------
ÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥ
```

Miscellaneous Tricks

Limit max subword size.

```
$ from transformers import AutoTokenizer
$ vocab = AutoTokenizer.from_pretrained("gpt2").vocab
$ [x for x in vocab if len(x) > 50]
```

yields:

```
=================================================================
................................................................
----------------------------------------------------------------
________________________________________________________________
================================================================
----------------------------------------------------------------
ÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤ
--------------------------------------------------------
ÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥÃĤÃĥ
```

Miscellaneous Tricks

```
$ from transformers import AutoTokenizer
$ tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

Miscellaneous Tricks

```
$ from transformers import AutoTokenizer
$ tokenizer = AutoTokenizer.from_pretrained("gpt2")



$ tokenizer.tokenize("1945")
1945
$ tokenizer.tokenize("1969")
1969
$ tokenizer.tokenize("1961")
19 61
```

Miscellaneous Tricks

```
$ from transformers import AutoTokenizer
$ tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
$ tokenizer.tokenize("1945")
1945
$ tokenizer.tokenize("1969")
1969
$ tokenizer.tokenize("1961")
19 61
```

```
$ tokenizer.tokenize("66666666")
66666666
$ tokenizer.tokenize("6666666")
6666 666
$ tokenizer.tokenize("00200000")
00200000
$ tokenizer.tokenize("00100000")
00 100 000
```

Miscellaneous Tricks

Don’t include numbers in subwords.

```
$ from transformers import AutoTokenizer
$ tokenizer = AutoTokenizer.from_pretrained("gpt2")
```

```
$ tokenizer.tokenize("1945")
1945
$ tokenizer.tokenize("1969")
1969
$ tokenizer.tokenize("1961")
19 61
```

```
$ tokenizer.tokenize("66666666")
66666666
$ tokenizer.tokenize("6666666")
6666 666
$ tokenizer.tokenize("00200000")
00200000
$ tokenizer.tokenize("00100000")
00 100 000
```

Miscellaneous Tricks

- Multilingual LM:
  - 1B English words
  - 10M Swiss German words
- Train BPE on 1010M words.

Miscellaneous Tricks

- Multilingual LM:
  - 1B English words
  - 10M Swiss German words
- Train BPE on 1010M words.

```
$ tokenizer("I see something that you don't.")
I see something that you don 't .

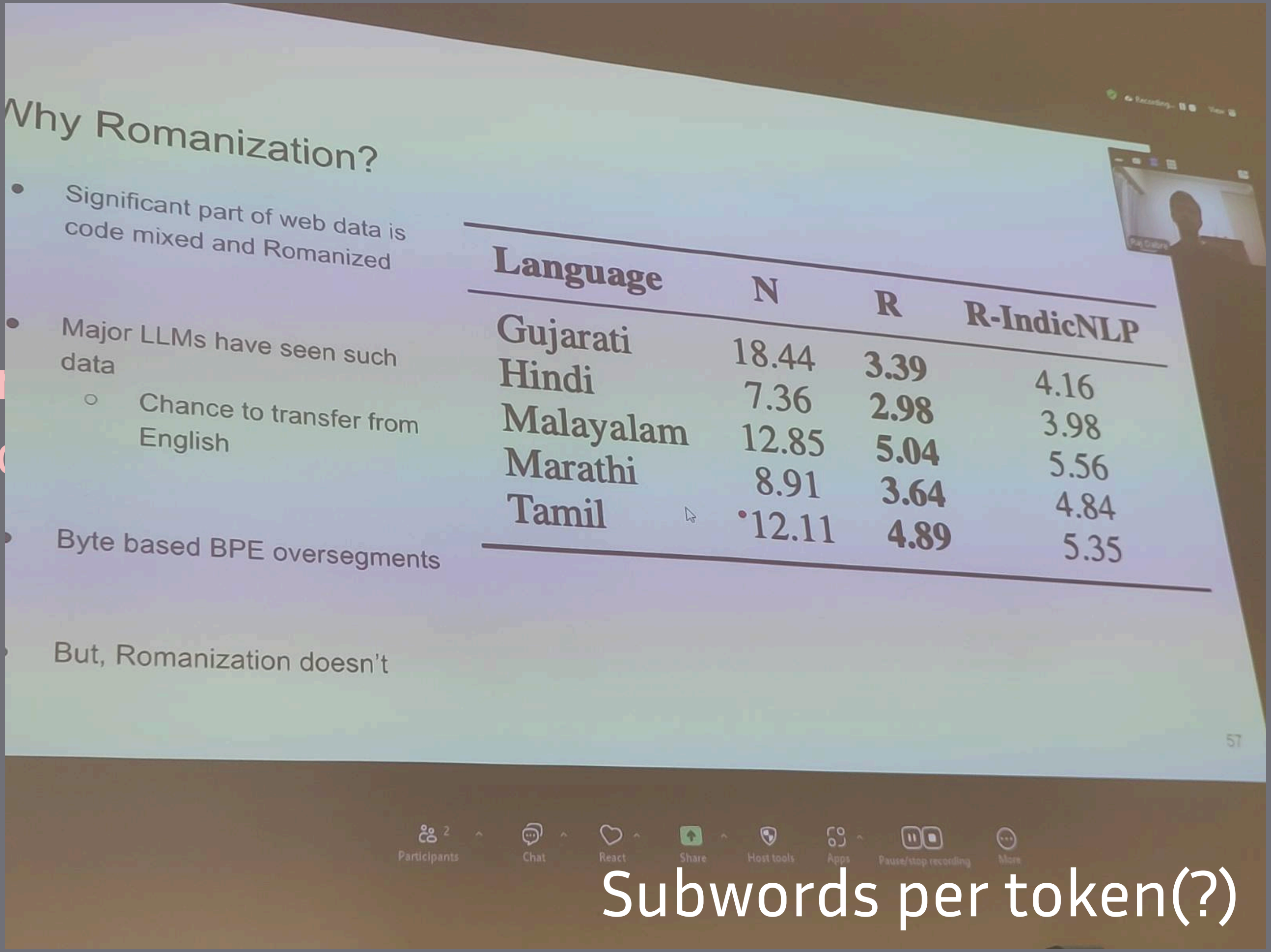
$ tokenizer("Ich sehe öpis, das de nöd siehst.")
I ch se he ö p is , das de n ö d s ie h st .
```

Miscellaneous Tricks

- Multilingual LM:
  - 1B English words
  - 10M Swiss German words
- Train BPE on 1010M words.

```
$ tokenizer("I see somethi
I see something that you d

$ tokenizer("Ich sehe öpis
I ch se he ö p is , das de
```



Subwords per token(?)

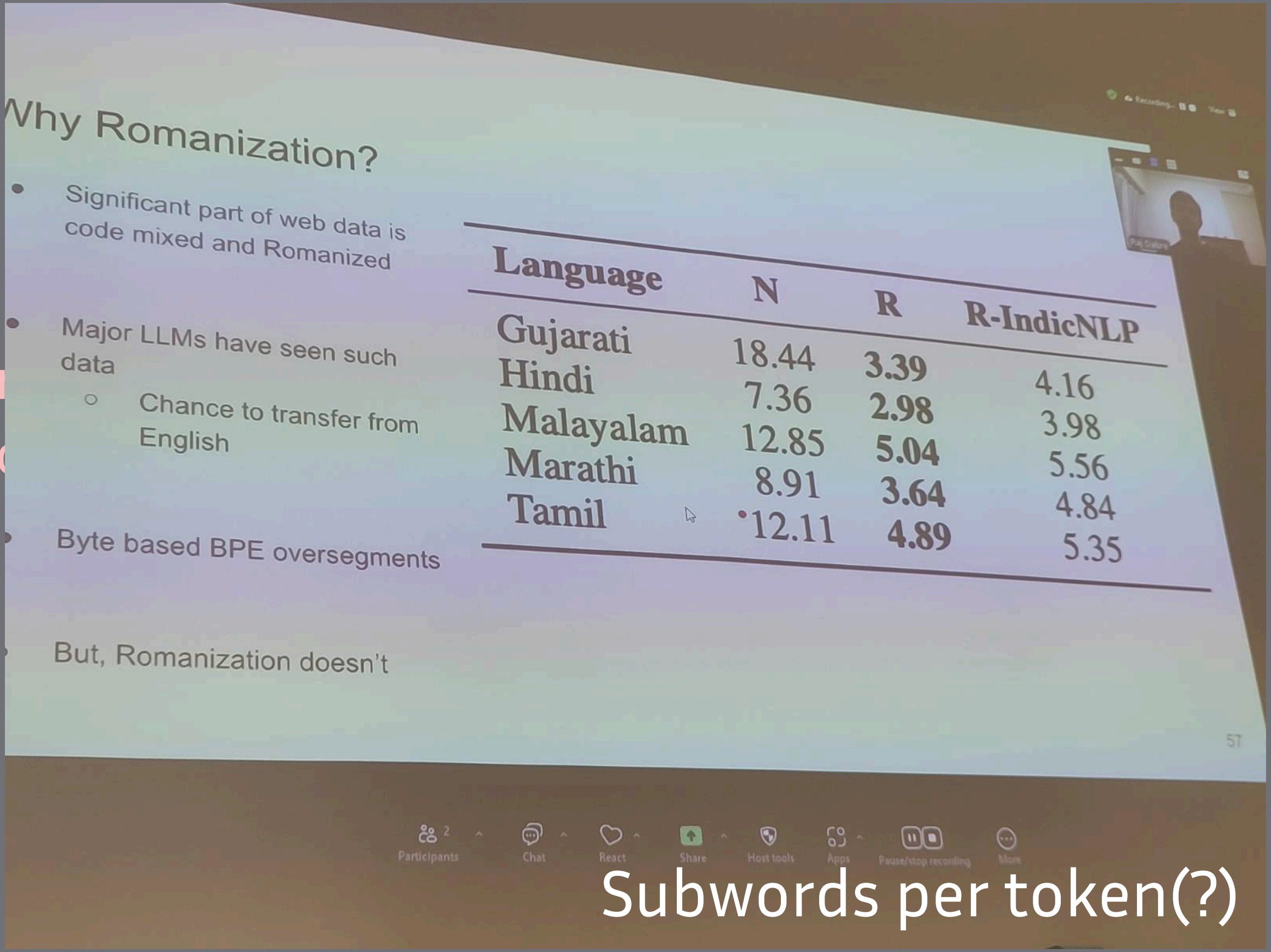
Miscellaneous Tricks

Supersample low-frequency data (if that's what you want).

- Multilingual LM:
  - 1B English words
  - 10M Swiss German words
- Train BPE on 1010M words.

```
$ tokenizer("I see somethi
I see something that you d

$ tokenizer("Ich sehe öpis
I ch se he ö p is , das de
```



Subwords per token(?)

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

$\overline{\mu} \leftarrow \langle\rangle$
for $i \in \{1, ..., M\}$ **do**
    $\mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
    $\overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
    $\overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
end for

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad\mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \ \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad\overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad\overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
6. **end for**

$\overline{x} =$ a a a a a b a b a b a

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \ \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

$\overline{x} =$ a a a a a b a b a b a

$\mathrm{Freq}(x, (\mu', \mu''))$ = ⟨ a a → 4, b a → 3, a b → 1, ⟩

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \text{argmax}_{\mu', \mu'' \in \overline{x}} \text{ Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \text{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

$\overline{x} =$ a a a a a b a b a b a

$\text{Freq}(x, (\mu', \mu''))$ = ⟨ a a → 4, b a → 3, a b → 1, ⟩

$\text{ApplyMerge}($a a $\rightarrow$ aa$, \overline{x}) =$ aa aa a b a b a b a

This code is wrong

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \text{argmax}_{\mu', \mu'' \in \overline{x}} \text{ Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \text{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle\mu\rangle$
6. **end for**

$\overline{x} =$ a a a a a b a b a b a

$\text{Freq}(x, (\mu', \mu''))$ = ⟨ a a → 4, b a → 3, a b → 1, ⟩

$\text{ApplyMerge}($a a $\rightarrow$ aa$, \overline{x}) =$ aa aa a b a b a b a

$|\overline{x}| - |\text{ApplyMerge}($a a $\rightarrow$ aa$, \overline{x})| = 2 \neq 4$

This code is wrong

Fix Freq computation.

Input: $\overline{x}$ (text to compress)

Output: $\overline{x}$ (compressed text) $\overline{\mu}$ (compression merges)

1. $\overline{\mu} \leftarrow \langle\rangle$
2. **for** $i \in \{1, ..., M\}$ **do**
3. $\quad \mu \leftarrow \mathrm{argmax}_{\mu', \mu'' \in \overline{x}} \mathrm{Freq}(x, (\mu', \mu''))$
4. $\quad \overline{x} \leftarrow \mathrm{ApplyMerge}(\mu, \overline{x})$
5. $\quad \overline{\mu} \leftarrow \overline{\mu} \circ \langle \mu \rangle$
6. **end for**

$\overline{x} =$ a a a a a b a b a b a

$\mathrm{Freq}(x, (\mu', \mu''))$ = ⟨ a a → 4, b a → 3, a b → 1, ⟩

$\mathrm{ApplyMerge}($a a $\rightarrow$ aa$, \overline{x}) =$ aa aa a b a b a b a

$|\overline{x}| - |\mathrm{ApplyMerge}($a a $\rightarrow$ aa$, \overline{x})| = 2 \neq 4$

A Formal Perspective on Byte-Pair Encoding **ACL 2023**

Tokenization and the Noiseless Channel **ACL 2023**

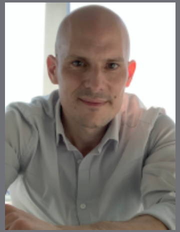
Two Counterexamples to Tokenization and the Noiseless Channel Cognetta et al. **LREC-COLING 2024**

Distributional Properties of Subword Tokenization Cognetta et al. **preprint 2024**

Marco Cognetta

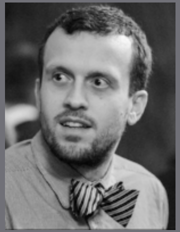
Mrinmaya Sachan

Ryan Cotterell



Juan Gastaldi

Clara Meister



Tim Vieira

Sangwhan Moon

Naoaki Okazaki

Leo Du

..and others

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal
- Think about distributions in stochastic tokenizations

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal
- Think about distributions in stochastic tokenizations

- High-entropy tokenization is good
- ```
  pip install tokenization-scorer
  tokenization-scorer -i data1.txt
  tokenization-scorer -i data2.txt
  ```
  ```

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal
- Think about distributions in stochastic tokenizations

- High-entropy tokenization is good
- `pip install tokenization-scorer`

```
tokenization-scorer -i data1.txt
tokenization-scorer -i data2.txt
```

What makes some tokenizations better than others?

Future

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal
- Think about distributions in stochastic tokenizations
- High-entropy tokenization is good
- `pip install tokenization-scorer`

```
tokenization-scorer -i data1.txt
tokenization-scorer -i data2.txt
```

Future

What makes some tokenizations better than others?

Multilingual tokenization?

- BPE is compression
- BPE is greedy & not optimal
- BPE is approximately optimal
- Think about distributions in stochastic tokenizations

- High-entropy tokenization is good
- `pip install tokenization-scorer`

```
tokenization-scorer -i data1.txt
tokenization-scorer -i data2.txt
```

What makes some tokenizations better than others?

Multilingual tokenization?

Future

Thank you

Rényi Entropy

$$H_\alpha(X) = \frac{1}{1-\alpha} \log\left(\sum_{i=1}^{n} p_i^\alpha\right)$$