Two Counterexamples to *Tokenization and the Noiseless Channel*

Marco Cognetta[⊡⋔] **Vilém Zouhar**[E] **Sangwhan Moon**[⊡⋔] **Naoaki Okazaki**[⊡⋔]

[⊡⋔]Tokyo Institute of Technology, School of Computing
[E]ETH Zurich, Department of Computer Science
`cognetta.marco@gmail.com, vzouhar@ethz.ch, sangwhan@iki.fi, okazaki@c.titech.ac.jp`

Abstract

In *Tokenization and the Noiseless Channel* (Zouhar et al., 2023a), Rényi efficiency is suggested as an intrinsic mechanism for evaluating a tokenizer: for NLP tasks, the tokenizer which leads to the highest Rényi efficiency of the unigram distribution should be chosen. The Rényi efficiency is thus treated as a predictor of downstream performance (e.g., predicting BLEU for a machine translation task), without the expensive step of training multiple models with different tokenizers. Although useful, the predictive power of this metric is not perfect, and the authors note there are additional qualities of a good tokenization scheme that Rényi efficiency alone cannot capture.

We describe two variants of BPE tokenization which can arbitrarily increase Rényi efficiency while decreasing the downstream model performance. These counterexamples expose cases where Rényi efficiency fails as an intrinsic tokenization metric and thus give insight for building more accurate predictors.

Keywords: machine-translation, tokenization, byte-pair-encoding, evaluation

1. Introduction

The tokenizer choice strongly impacts the downstream NLP model performance (Domingo et al., 2019). At the same time, it is difficult to select the best one, because comparing two tokenizers typically requires fully training a model on top of each tokenizer, which can be prohibitively expensive, taking days or weeks of compute times. This motivates the search for an intrinsic evaluation of tokenizers — finding easy-to-compute metrics that can be evaluated using only the tokenized text and which signal if the tokenization will be good for the downstream task (see Figure 1).

In *Tokenization and the Noiseless Channel*, Zouhar et al. (2023a) propose a new metric based on a generalization of Shannon Entropy — Rényi efficiency of the unigram distribution. This quantity is slightly better correlated to downstream model performance (BLEU score on a translation task) than a percentile frequency metric, which they adopted based on Gowda and May (2020). It is also much better correlated than Shannon Entropy (Shannon, 1948) and average sequence length. The authors suggest a characteristic of good tokenizers: not only should a tokenizer contain few rare tokens (tokens that appear infrequently in tokenized texts), but they should also not contain many very high-frequency tokens. While they propose this metric for intrinsically evaluating tokenizers before training, they note that it does not fully account for the relative performances of tokenizers and leave the search for other metrics as an open problem. This means there is some variance in the performances of all possible tokenizers which the Rényi efficiency does not capture.
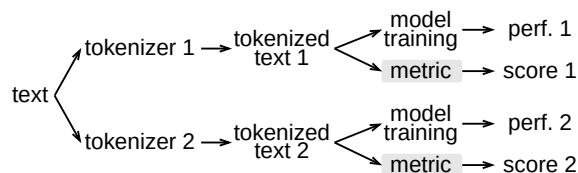
In this work, we introduce two variants of BPE to-

Figure 1: The same model is trained on two different tokenizations. We are looking for metric such that, across multiple settings, *score 1 > score 2* if and only if *performance 1 > performance 2*.

kenizers for which we can explicitly increase Rényi efficiency while degrading the downstream model performance. This goes against the aforementioned hypothesis where high Rényi efficiency implies better expected performance. We hope that these examples will help spur more research into where Rényi efficiency and other related metrics accurately predict performance and where they fail. Ultimately, these efforts are intended to create a better predictor of downstream performance for NLP tasks based on the tokenized text.

Structure:

2) Introduction to Byte-Pair Encoding.
3) Rényi efficiency, other metrics and their respective tokenization hypotheses.
4) Two families of counterexamples to the Rényi efficiency tokenization hypothesis and proofs.
5) Empirical verification of counterexamples with machine translation performance[1].

---

[1]Code to reproduce all experiments:
github.com/mcognetta/TokenizationNoiselessCounterexamples

2. Byte-Pair Encoding

Byte-Pair Encoding was proposed as a compression algorithm (Gage, 1994), repurposed in NLP for machine translation (Sennrich et al., 2016), and later formalized and bounded mathematically (Zouhar et al., 2023b). The BPE algorithm has two parts: (1) training, where the merges are constructed, and (2) tokenization, where the merges are applied to new text. In this paper, we focus only on the *tokenization* side and we use the standard BPE *training* algorithm for all experiments.

BPE Training. A trained BPE tokenizer is a triplet $(\Sigma, \mathcal{V}, \boldsymbol{\mu})$, where Σ is a set of *atomic* characters, $\mathcal{V} \subset \Sigma^+$ is the *subword vocabulary*, and $\boldsymbol{\mu} \in \mathcal{V} \times \mathcal{V}$ is an ordered list of *merges*. The $\boldsymbol{\mu}$ defines how subwords combine to form new subwords. We use the notation of APPLY($[a, b], t$) to *apply* a merge to a sequence – that is, to replace each instance of consequent symbols ab with ab in the sequence. See examples of notation:

Σ	a, b, c, d, ...
$\mathcal{V}$	a, the, cow, token, -ization, ...
$\boldsymbol{\mu}$	[t,h], [[t,h],e], ...
APPLY([c, -o], c -o -w) → co -w, ...	

We construct a BPE tokenizer using the standard algorithm described by Gage (1994); Sennrich et al. (2016); Zouhar et al. (2023b). In our notation, $\boldsymbol{\mu}$ is ordered by the order in which a merge was added to the BPE vocabulary during construction. The first element of $\boldsymbol{\mu}$ is the first pair that merged during BPE initialization, and we proceed in decreasing order from there. To see how the merge sequence $\boldsymbol{\mu}$ is constructed. For completeness, the BPE *training* algorithm is provided in Appendix A.

BPE Tokenization. We describe the standard BPE tokenization procedure in Algorithm 1, which serves as the baseline. The BPE tokenization algorithm is applied independently on all words in the corpus, and so Algorithm 1 describes an algorithm that takes a single word (as a sequence of characters from Σ) as input along with the BPE tokenizer. We modify this algorithm and its inputs for our experiments later. In words, the BPE tokenization algorithm starts with a sequence of characters and iteratively merges the first pair in $\boldsymbol{\mu}$, then the second and ultimately the last.

3. Rényi Efficiency and Other Metrics

The performance-from-tokenization prediction task can be formalized as finding a scalar metric m over a sequence of tokens from $\mathcal{V}^+$, therefore $m : \mathcal{V}^+ \rightarrow \mathbb{R}$. The goal for this metric is to correlate with

---

Algorithm 1: Standard BPE tokenization
Inputs: word $w \in \Sigma^*$, merges $\boldsymbol{\mu}$,
$\qquad \Sigma \cup \bigcup_{[a,b] \in \boldsymbol{\mu}} \{ab\} = \mathcal{V}$
Output: tokenized sequence $t \in \mathcal{V}^*$

1: $t \leftarrow$ CHARACTERSEQUENCE(w)
2: **for** $[a, b] \in \boldsymbol{\mu}$ **do**
3: $\quad t \leftarrow$ APPLY($[a, b], t$)
4: **end for**
5: **return** t

---

some downstream performance, such as BLEU of a machine translation system.

Baseline. The simplest metric, denoted C, is to use the average number of tokens per line. The hypothesis is that given the same vocabulary size, if the tokenized text contains fewer tokens, then the tokenization is likely better. In the following example, the first tokenization is likely better because the text is tokenized into fewer pieces.

C(the quick fox jump -ed) $= 5$
C(th -e br -own fox j -u -m -ped) $= 9$

With a large enough vocabulary size, one could represent each word as a separate token. This tokenization would obtain the lowest possible score by C and be marked as the best one, even though it would yield very poor downstream task results because of the lack of generalization to unseen words (Sennrich et al., 2016).

Percentile Frequency. For more nuanced metrics, we turn to the unigram distribution of the tokenized text, $p_{\mathcal{V}}$. For example $p_{\mathcal{V}}(\text{"the"}) = 0.02$. Given the start and end percentiles γ_1 and γ_2, the quantity is defined as:

$$F_{\gamma_1, \gamma_2}(\mathcal{V}) = \sum_{w \in \text{PERC.}_{p_{\mathcal{V}}}(\gamma_1, \gamma_2)} p_{\mathcal{V}}(w), \quad (1)$$

and the higher the sum, the better the tokenizer should be (Gowda and May, 2020). We use $F_{0.03, 0.83}$ based on (Zouhar et al., 2023a), which performed a hyperparameter scan to determine the percentiles which best correlated with downstream performance.

Rényi Efficiency. Finally, we bring to attention the Rényi entropy of the random variable $W_{\mathcal{V}}$ distributed according to $p_{\mathcal{V}}$ and its associated quantity, Rényi efficiency:

$$H_\alpha(W_{\mathcal{V}}) = \lim_{\alpha' \to \alpha} \frac{1}{1 - \alpha'} \log \left(\sum_{w \in \mathcal{V}} p_{\mathcal{V}}(w)^{\alpha'} \right) \quad (2)$$

$$\text{Eff}_\alpha(W_{\mathcal{V}}) \cong \frac{H_\alpha(W_{\mathcal{V}})}{\log |\mathcal{V}|} \quad (3)$$

The higher the Rényi entropy, the more "balanced" the unigram distribution is, meaning there is a less pronounced difference between the frequencies of the most common and least common tokens. We use the Rényi efficiency instead of Rényi entropy to normalize the effect of vocabulary size. This works similarly with Shannon entropy and Shannon efficiency (when $\alpha = 1$). This metric is found by Zouhar et al. (2023a) to correlate the highest with the downstream performance on a translation task.

4. Counterexamples

The Rényi Efficiency tokenization hypothesis states that the higher the Rényi efficiency of the unigram distribution, the better the downstream model performance. To provide counterexamples, we want to find tokenizers that increase the Rényi efficiency while worsening the performance. We provide two such counterexamples: RANDOM-DROP BPE and DUPLICATE BPE. In the Appendix we will also include a naïve method which can arbitrarily *lower* the Rényi efficiency of a tokenizer (by inflating $\mathcal{V}$) while not affecting the performance, thus also serving as a counterexample.

4.1. RANDOM-DROP BPE

Given a BPE tokenizer $(\Sigma, \mathcal{V}, \boldsymbol{\mu})$, we form a new tokenizer by selecting integer hyperparameters $1 \leq k \leq N \leq |\mathcal{V}|$. Then, we randomly select k subwords from the top N most frequent subwords in $\mathcal{V}$ (restricted to non-atomic subwords) to form a set $\mathcal{D}$. Setting $N = |\mathcal{V}|$ would allow any non-atomic token to be a candidate for inclusion in $\mathcal{D}$. A RANDOM-DROP BPE tokenizer is then defined as $(\Sigma, \mathcal{V}, \boldsymbol{\mu}, \mathcal{D})$. The elements of $\mathcal{D}$ are marked for decomposition via a function DEC. We omit the parameterization of DEC on $(\Sigma, \mathcal{V}, \boldsymbol{\mu}, \mathcal{D})$ for brevity.

$$\text{DEC}(z) = \begin{cases} \text{DEC}(x) \ \text{DEC}(y), & z = [x, y] \wedge z \in \mathcal{D} \\ z, & \text{otherwise} \end{cases} \quad (4)$$

Given a subword $z \in \mathcal{D}$, DEC recursively undoes merges, splitting tokens into their merge pairs, until none of the tokens remaining in the sequence are in $\mathcal{D}$. This decomposition is applied *after* the regular BPE tokenization is complete. See an example of the two-stage process in Figure 2. The RANDOM-DROP BPE tokenizer inference is implemented by replacing Line 5 of Algorithm 1 with:

$$\textbf{return} \ \langle \text{DEC}(t_i) \mid 1 \leq i \leq |t| \rangle \quad (5)$$

This tokenizer modification is distinct from BPE-Dropout regularization (Provilkov et al., 2020). RANDOM-DROP BPE is entirely deterministic (the set $\mathcal{D}$ is selected after vocabulary construction and

is never changed), and so tokenizing a sequence multiple times will always result in the same sequence. Also, the decomposition happens after the standard BPE tokenization. In contrast, BPE-Dropout is performed during tokenization at each iteration of the merging algorithm by selectively blocking merges. Given the same input, BPE-Dropout can produce different final tokenizations.
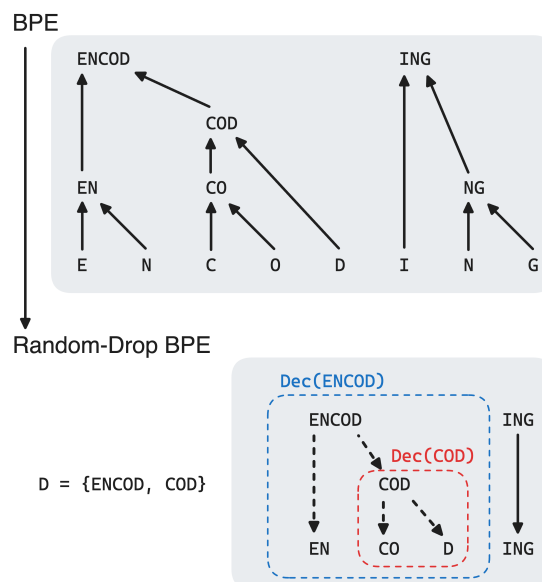

Figure 2: An example of RANDOM-DROP BPE with $\mathcal{D} = \{$ENCOD, COD$\}$. The tokenization process is two-staged: (1) a regular BPE tokenizer is used to produce a tokenization, and (2) RANDOM-DROP BPE recursively decomposes subwords that appear in the set $\mathcal{D}$ (first ENCOD, and then COD). Since ING is not in $\mathcal{D}$, it is unchanged during the decomposition step. The final tokenized text is EN -CO -D -ING. If $\mathcal{D}$ was just $\{$ENCOD$\}$, then the final tokenization would have been EN -COD -ING, and if it was just $\{$COD$\}$, the final tokenization would have been ENCOD -ING, since -COD was not in the final tokenization produced by the initial BPE pass.

Efficiency Increase. In all of our experiments (Section 5), the RANDOM-DROP BPE tokenizer improves Rényi efficiency compared to its underlying BPE tokenizer. Now, we give some intuition for why this should happen empirically — specifically, in the form of a proof of the sufficient conditions for a token to, if dropped, increase the Rényi *entropy*. The condition does not rely on the vocabulary size, and so does not relate directly to Rényi *efficiency*, which introduces a divisor of $\log(|\mathcal{V}|)$. Under the assumption that all subwords in the vocabulary appear at least once in the final tokenized corpus,[2] increasing Rényi entropy always increases Rényi

---

[2]A subword can be a valid merge but not appear in any final tokenization if it always appears as a part of a longer subword. In this case, RANDOM-DROP BPE

efficiency, because we are effectively removing one token from the vocabulary and the normalizing factor is reduced to $\log(|\mathcal{V}| - 1)$.

Theorem 4.1. *Let $c_{\mathcal{V}}(w)$ be the unigram frequency of $w \in \mathcal{V}$, and $T_{\mathcal{V}} = \sum_{w \in \mathcal{V}} c(w)$ so that $p_{\mathcal{V}}(w) = \frac{c_{\mathcal{V}}(w)}{T_{\mathcal{V}}}$ is the unigram distribution. Suppose subword $x = (y, z) \in \mathcal{V}$ is chosen for decomposition, resulting in a new vocabulary $\mathcal{V}' = \mathcal{V} \setminus \{x\}$. Let $W_{\mathcal{V}}$ and $W_{\mathcal{V}'}$ be the corresponding random variables and let $\alpha > 1$ be the parameter for Rényi entropy. Then $H_\alpha[W_{\mathcal{V}}] < H_\alpha[W_{\mathcal{V}'}]$ if*

$$\left(1 + \frac{c_{\mathcal{V}}(x)}{T_{\mathcal{V}}} \right)^\alpha > 1$$
$$+ ((c_{\mathcal{V}}(x) + c_{\mathcal{V}}(y))^\alpha + (c_{\mathcal{V}}(x) + c_{\mathcal{V}}(z))^\alpha$$
$$- c_{\mathcal{V}}(x)^\alpha - c_{\mathcal{V}}(y)^\alpha - c_{\mathcal{V}}(z)^\alpha)/\left(\sum_{w \in \mathcal{V}} c_{\mathcal{V}}(w)^\alpha \right)$$

4.2. DUPLICATION BPE

Zouhar et al. (2023a) sought to find the Rényi Entropy parameter α that was best correlated with the downstream task performance. In their setup, the best $\alpha = 2.7$ but $\alpha = 3$, the closest integer, is the default in their released toolkit and we use it for all our experiments. Because this value is greater than 1, it suggests that not only the existence of many very low-frequency tokens hurt performance, but also that having very high-frequency tokens can degrade the performance.

The elimination of rare tokens seems straightforward. If they are not single-character (atomic), they can be replaced with their constituent parts. For example markup → mark -up. In some cases this decomposition can increase the frequency of an already-high-frequency token, such as again → a -gain, which would increase the frequency of the already common subword a. In addition, replacing all instances of the high-frequency token will simply add its frequency to two other subwords, making them both appear at least as frequently as the now-eliminated token. We propose a way to eliminate high-frequency tokens while avoiding these issues.

Given a BPE tokenizer $(\Sigma, \mathcal{V}, \boldsymbol{\mu})$, we select high-frequency tokens and replace them with tokens that have identical surface form (i.e., they spell the same thing) but with different indices (i.e., they have different embeddings in the neural model). For presentation purposes, we modify the surface form of the token to disambiguate them. For example, given a high-frequency subword -ing, we denote the duplicates -ing$_1$, -ing$_2$, ..., -ing$_k$.

Formally, a DUPLICATION BPE tokenizer is formed by choosing integers $1 \le N \le |\mathcal{V}|$ and

---

can increase the vocabulary size if a specific drop is performed where both component subwords have 0 frequency. If the increase in entropy is too small, the larger normalizing factor can decrease the efficiency.

$2 \le k$. For each of the top N most frequent tokens (measured by their frequency in the corpus), denoted by $\mathcal{X}$, we add k duplicated tokens to $\mathcal{V}$. Then, during BPE inference, in the final tokenized sequence of a word, if any of the N duplicated subwords appear, we uniformly at random replace it with one of the k duplicates.

$$\text{SUB}(z) = \begin{cases} z' \sim \mathcal{U}(\{z_i\}_{1 \le i \le k}) & z \in \mathcal{X} \\ z & \text{otherwise} \end{cases} \quad (16)$$

Where $\mathcal{U}$ is the discrete uniform distribution (each of the k duplicates have the same probability). DUPLICATION BPE inference is implemented by replacing Line 5 of Algorithm 1 with

$$\textbf{return } \langle \text{SUB}(t_i) \mid 1 \le i \le |t| \rangle \quad (17)$$

During the downstream evaluation (e.g., BLEU), we renormalize the tokens by replacing all duplicated tokens with their original surface form (i.e., reverting -ing$_3$ to -ing). Detokenization and evaluation is then done as normal. See an example of this modification in Example 1.

Explanation. We are eliminating high-frequency subwords by spreading their probability mass across the duplicate tokens. On the other hand, the downstream model performance should be bounded above by the baseline due to the fact that the model is not aware that the tokens are duplicates of each other and will learn a distribution that spreads the probability mass between them. In particular, during beam search, the pessimal case is that the cumulative probability mass of the duplicates is higher than any other token, but individually, they are all below the threshold to be included in the beam.

$\mathcal{X}$	Tokenization
Ø (original)	the apple the berry the citrus the dill the elderberry
{the, apple}	the$_3$ apple$_4$ the$_5$ berry the$_4$ citrus the$_3$ dill the$_7$ elderberry

Example 1: DUPLICATION BPE with $k{=}10$ (second row) changes the token but the surface form remains the same (apple$_4$ = apple$_3$).

Rényi Entropy. Formally, we treat the algorithm as modifying repeatedly some element w_x and changing its frequency from $p_{\mathcal{V}}(w_x)$ to $\frac{p_{\mathcal{V}}(w_x)}{k}$. It is easy to see that for $k = 1$, the algorithm does not change the unigram distribution. Therefore, for the following two theorems, assume $k > 1$.

Theorem 4.2. *Let $p_{\mathcal{V}}$ be the unigram distribution of a tokenizer and let $p'_{\mathcal{V}}$ be the DUPLICATION BPE unigram distribution with $k > 1$. Let $W_{\mathcal{V}}$ and $W'_{\mathcal{V}}$*

Let $\alpha > 1$. When removing x from the vocabulary, every instance of x in the tokenized sequence gets replaced with (y, z). Thus, in $\mathcal{V}'$, y and z's counts are increased by $c_{\mathcal{V}}(x)$ and the overall total number of tokens is increased by $c_{\mathcal{V}}(x)$ (as the total change is $2c_{\mathcal{V}}(x) - c_{\mathcal{V}}(x)$), and so $T_{\mathcal{V}'} = T_{\mathcal{V}} + c_{\mathcal{V}}(x)$, $c_{\mathcal{V}'}(x) = 0$, $c_{\mathcal{V}'}(y) = c_{\mathcal{V}}(x) + c_{\mathcal{V}}(x)$, $c_{\mathcal{V}'}(z) = c_{\mathcal{V}}(x) + c_{\mathcal{V}}(z)$, and $c_{\mathcal{V}'}(w) = c_{\mathcal{V}}(w)$ for all other $w \in \mathcal{V}\backslash\{x, y, z\}$. Then:

$$H_\alpha[W_{\mathcal{V}}] < H_\alpha[W'_{\mathcal{V}}] \quad (6) \qquad \frac{1}{1-\alpha}\log\left(\sum_{w\in\mathcal{V}} p_{\mathcal{V}}(w)^\alpha\right) < \frac{1}{1-\alpha}\log\left(\sum_{w\in\mathcal{V}} p_{\mathcal{V}'}(w)^\alpha\right) \quad (7)$$

$$\sum_{w\in\mathcal{V}} p_{\mathcal{V}}(w)^\alpha > \sum_{w\in\mathcal{V}} p_{\mathcal{V}'}(w)^\alpha \quad (8) \qquad \frac{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha}{T_{\mathcal{V}}^\alpha} > \frac{\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha}{T_{\mathcal{V}'}^\alpha} \quad (9)$$

$$\left(\frac{T_{\mathcal{V}'}}{T_{\mathcal{V}}}\right)^\alpha > \frac{\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} \quad (10) \qquad \left(\frac{T_{\mathcal{V}'}}{T_{\mathcal{V}}}\right)^\alpha = \left(\frac{T_{\mathcal{V}} + c_{\mathcal{V}}(x)}{T_{\mathcal{V}}}\right)^\alpha = \left(1 + \frac{c_{\mathcal{V}}(x)}{T_{\mathcal{V}}}\right)^\alpha > \frac{\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} \quad (11)$$

In Equation (11) the left-hand side is in the form we want. The right-hand-side numerator, $\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha$, can be rewritten as:

$$\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha = \left(\sum_{\substack{w\in\mathcal{V}\\\backslash\{x,y,z\}}} c_{\mathcal{V}}(w)^\alpha\right) + c_{\mathcal{V}'}(y)^\alpha + c_{\mathcal{V}'}(z)^\alpha = \left(\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha\right) + c_{\mathcal{V}'}(y)^\alpha + c_{\mathcal{V}'}(z)^\alpha - c_{\mathcal{V}}(x)^\alpha - c_{\mathcal{V}}(y)^\alpha - c_{\mathcal{V}}(z)^\alpha \quad (12)$$

Noting that $c_{\mathcal{V}'}(y) = c_{\mathcal{V}}(x) + c_{\mathcal{V}}(y)$ and $c_{\mathcal{V}'}(z) = c_{\mathcal{V}}(x) + c_{\mathcal{V}}(z)$ and that $\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha$ is the right-hand-side denominator of Equation (11). We arrive at our final right-hand side:

$$\frac{\sum_{w\in\mathcal{V}'} c_{\mathcal{V}'}(w)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} = 1 + \frac{c_{\mathcal{V}'}(y)^\alpha + c_{\mathcal{V}'}(z)^\alpha - c_{\mathcal{V}}(x)^\alpha - c_{\mathcal{V}}(y)^\alpha - c_{\mathcal{V}}(z)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} \quad (13)$$

$$= 1 + \frac{(c_{\mathcal{V}}(x) + c_{\mathcal{V}}(y))^\alpha + (c_{\mathcal{V}}(x) + c_{\mathcal{V}}(z))^\alpha - c_{\mathcal{V}}(x)^\alpha - c_{\mathcal{V}}(y)^\alpha - c_{\mathcal{V}}(z)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} \quad (14)$$

Combining the two sides of the equation gives our final result:
$$\left(1 + \frac{c_{\mathcal{V}}(x)}{T_{\mathcal{V}}}\right)^\alpha > 1 + \frac{(c_{\mathcal{V}}(x) + c_{\mathcal{V}}(y))^\alpha + (c_{\mathcal{V}}(x) + c_{\mathcal{V}}(z))^\alpha - c_{\mathcal{V}}(x)^\alpha - c_{\mathcal{V}}(y)^\alpha - c_{\mathcal{V}}(z)^\alpha}{\sum_{w\in\mathcal{V}} c_{\mathcal{V}}(w)^\alpha} \quad (15)$$

Proof of Theorem 4.1 — Rényi Entropy increases from RANDOM-DROP BPE.

be the corresponding random variables. Then $H[W_{\mathcal{V}}] < H[W'_{\mathcal{V}}]$.

Theorem 4.3. *Let $p_{\mathcal{V}}$ be the unigram distribution of a tokenizer and let $p'_{\mathcal{V}}$ be the DUPLICATION BPE unigram distribution with $k > 1$. Let $W_{\mathcal{V}}$ and $W'_{\mathcal{V}}$ be the corresponding random variables. Then, $H_\alpha[W_{\mathcal{V}}] < H_\alpha[W'_{\mathcal{V}}]$.*

An illustration of how one application affects the Shannon and Rényi entropies is in Example 2.

k	Unigram Distribution	H	$H_{0.5}$	H_3	Eff	$Eff_{0.5}$	Eff_3
-	(0.4, 0.3, 0.2, 0.1)	1.85	1.92	1.66	1.33	1.38	1.20
2	(0.2, 0.2, 0.3, 0.2, 0.1)	2.25	2.28	2.13	1.40	1.42	1.33
10	(0.04, ..., 0.04, 0.3, ...)	3.18	3.45	2.39	1.24	1.35	0.93

Example 2: Entropy/efficiency change after applying DUPLICATION BPE to the first element in the original sequence (first row) which is expanded to two or ten elements. Note the case where efficiency decreases, despite entropy increase, which is an artifact of the small initial vocabulary size.

Rényi Efficiency. We do not have a formal proof that this algorithm increases efficiency because of the change to the denominator $\log|\mathcal{V}|$ in Equation (2). In fact, we show a counterexample where

this algorithm *decreases* the efficiency in Example 2. When the duplication factor k is high and the vocabulary small, the size of the resulting vocabulary grows too quickly. Even if **H** grows (as it does in every example), $\log(|\mathcal{V}'|) = \log(|\mathcal{V}| + (k-1)N)$ grows faster and **Eff** decreases. For realistic vocabulary sizes, $\geq$ 4k, the relative increase in vocabulary size is much smaller and so **H** increasing tends to imply that **Eff** will also increase. We later (Tables 1 and 2) show empirically that for larger vocabularies the efficiency increases.

5. Experiments

We use the same MT Transformer model as Zouhar et al. (2023a), transformer-iwslt in fairseq, and the iwslt14 German→English corpus for training.[3]

We do not use a joint vocabulary between the source and target languages so that we can tune them both independently. Through hyperparameter search, we found that the 6k/6k BPE model performed the best among all BPE vocabulary sizes (even with different source and target sizes), and we use it as a baseline. We also experiment with 4k/4k, 9k/9k, and 14k/14k baselines to demonstrate that our results hold across a variety of hyper-

---

[3]This is different than the dataset used by Zouhar et al. (2023a), but we chose to use a smaller corpus due to the computational costs.

Assume the algorithm modifies the frequency of token w_x.

$$H[W'_\mathcal{V}] = -\sum_{w \in \mathcal{V}} p_\mathcal{V}(w) \log(p_\mathcal{V}) + p_\mathcal{V}(w_x) \log(p_\mathcal{V}(w_x)) - k \frac{p_\mathcal{V}(w_x)}{k} \log\left(\frac{p_\mathcal{V}(w_x)}{k}\right) \tag{18}$$

$$= H[W_\mathcal{V}] + p_\mathcal{V}(w_x)\left(\log(p_\mathcal{V}(w_x)) - \log\left(\frac{p_\mathcal{V}(w_x)}{k}\right)\right) = H[W_\mathcal{V}] + p_\mathcal{V}(w_x) \log(k) > H[W_\mathcal{V}] \tag{19}$$

Proof of Theorem 4.2 — Shannon Entropy increases from DUPLICATION BPE.

Assume the algorithm modifies the frequency of token w_x. For $\alpha > 1$: $H_\alpha[W_\mathcal{V}] < H_\alpha[W'_\mathcal{V}]$.

$$\frac{1}{1-\alpha} \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha\right) < \frac{1}{1-\alpha} \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha\right) \tag{20}$$

$$\log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha\right) > \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha\right) \quad \textit{(from } \alpha > 1\textit{)} \tag{21}$$

$$\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha > \sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha \tag{22}$$

$$0 > -p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha \tag{23}$$

$$0 > p_\mathcal{V}(w_x)^\alpha \cdot (k^{1-\alpha} - 1) \tag{24}$$

$$1 > k^{1-\alpha} \quad \textit{(holds from } \alpha > 1 \text{ and } k > 1\textit{)} \tag{25}$$

The proof follows by reversing the order of equivalent inequalities. Conversely, for $\alpha < 1$: $H_\alpha[W_\mathcal{V}] < H_\alpha[W'_\mathcal{V}]$.

$$\frac{1}{1-\alpha} \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha\right) < \frac{1}{1-\alpha} \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha\right) \tag{26}$$

$$\log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha\right) < \log\left(\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha\right) \quad \textit{(from } \alpha < 1\textit{)} \tag{27}$$

$$\sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha < \sum_{w \in \mathcal{V}} p_\mathcal{V}(w)^\alpha - p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha \tag{28}$$

$$0 < -p_\mathcal{V}(w_x)^\alpha + k \cdot \left(\frac{p_\mathcal{V}(w_x)}{k}\right)^\alpha \tag{29}$$

$$0 < p_\mathcal{V}(w_x)^\alpha \cdot (k^{1-\alpha} - 1) \tag{30}$$

$$1 < k^{1-\alpha} \quad \textit{(holds from } \alpha < 1 \text{ and } k > 1\textit{)} \tag{31}$$

The proof follows by reversing the order of equivalent inequalities. For $\alpha = 1$ the Theorem 4.2 applies. This covers all cases of admissible $\alpha > 0$ and therefore concludes the proof.

Proof of Theorem 4.3 — Rényi Entropy increases from DUPLICATION BPE.

parameters and vocabulary sizes. For both of our proposed tokenizer families, we use the baseline BPE tokenizers as the starting point.

All reported predictor metrics were computed using `tokenization-scorer`[4] in their default settings and on the concatenated source+target tokenized training corpora. In particular, we use $\alpha = 3$ for all Rényi efficiency metrics.

5.1. RANDOM-DROP BPE

For this family, for each baseline BPE tokenizer (each of the baseline vocabulary sizes), we experimented with four hyperparameter settings. We varied the range of tokens that were candidates for being chosen for decomposition (N) with two settings: 2000 and $|\mathcal{V}|$. We also varied the number of

tokens that were chosen to be marked for decomposition (k) with two settings: 500 and 1000. For each configuration, we created three models using different seeds to choose $\mathcal{D}$. These three models have the same vocabulary size, but have different $\mathcal{D}$ and therefore produce different tokenizations. We train each configuration with three different seeds and take take the average BLEU score (of all nine models) as the representative score (**Overall** in Table 1). To mimic a more real-life setup, we also include the models with the best average scores for each configuration (**Best** in Table 1).

Table 1 contains the experimental results for this family. Across nearly every experiment, the RANDOM-DROP BPE tokenizers increase Rényi efficiency compared to their baseline BPE tokenizers *and* the average BLEU scores are higher. In one case, (6k/6k, $N = 2000$, $k = 1000$), the

---

[4]github.com/zouharvi/tokenization-scorer

Tokenizer	N	k	Overall		Best	
			Eff_α	BLEU	Eff_α*	BLEU*
BASELINE (4K/4K)	-	-	0.474	33.74	-	-
RANDOM-DROP (4k/4k)	2k	500	0.500	33.39	0.504	33.48
	2k	1k	0.474	32.76	0.483	32.89
	4k	500	0.497	33.72	0.498	33.85
	4k	1k	0.506	33.40	0.518	33.48
RANDOM-DROP (4.5k/4.5k)	2k	500	0.491	33.35	0.495	33.37
	4.5k	500	0.485	33.69	0.487	33.81
BASELINE (6K/6K)	-	-	0.444	33.94	-	-
RANDOM-DROP (6k/6k)	2k	500	0.468	33.46	0.471	33.46
	2k	1k	0.441	32.86	0.445	33.03
	6k	500	0.458	33.69	0.458	33.94
	6k	1k	0.473	33.60	0.472	33.71
RANDOM-DROP (6.5k/6.5k)	2k	500	0.462	33.37	0.464	33.44
	6.5k	500	0.451	33.69	0.453	33.70
BASELINE (9K/9K)	-	-	0.418	33.60	-	-
RANDOM-DROP (9k/9k)	2k	500	0.441	33.35	0.440	33.45
	2k	1k	0.426	32.73	0.440	32.87
	9k	500	0.426	33.68	0.425	33.74
	9k	1k	0.435	33.59	0.436	33.66
RANDOM-DROP (9.5k/9.5k)	2k	500	0.440	33.31	0.440	33.33
	9.5k	500	0.423	33.60	0.423	33.71
BASELINE (14K/14K)	-	-	0.394	33.59	-	-
RANDOM-DROP (14k/14k)	2k	500	0.415	33.05	0.417	33.09
	2k	1k	0.409	32.63	0.416	32.84
	14k	500	0.399	33.41	0.399	33.54
	14k	1k	0.405	33.35	0.404	33.45
RANDOM-DROP (14.5k/14.5k)	2k	500	0.412	33.11	0.415	33.17
	14.5k	500	0.396	33.49	0.396	33.54

Table 1: Experimental results for the RANDOM-DROP. For each BASELINE tokenizer, we compare to RANDOM-DROP (based on the respective BASELINE) with various hyperparameters. **Overall** and **Best** are based on three tokenizer seeds.

Tokenizer	N	k	Eff_α	BLEU
BASELINE (4K/4K)	-	-	0.474	33.74
DUPLICATION (4K/4K)	100	3	0.594	32.37
	100	5	0.648	31.32
	500	3	0.583	32.26
	500	5	0.627	N/A
BASELINE (6K/6K)	-	-	0.444	33.94
DUPLICATION (6K/6K)	100	3	0.560	32.27
	100	5	0.612	31.60
	500	3	0.552	32.43
	500	5	0.598	30.57
BASELINE (9K/9K)	-	-	0.418	33.60
DUPLICATION (9K/9K)	100	3	0.530	32.48
	100	5	0.581	31.64
	500	3	0.525	32.42
	500	5	0.572	30.93
BASELINE (14K/14K)	-	-	0.394	33.59
DUPLICATION (14K/14K)	100	3	0.501	32.49
	100	5	0.551	31.45
	500	3	0.498	32.55
	500	5	0.545	30.70

Table 2: Experimental results for the DUPLICATION. For each BASELINE tokenizer, we compare to DUPLICATION (based on the respective BASELINE) with various hyperparameters. The average over three training runs is used for all experiments. N/A: This configuration achieved less than 5 BLEU across several trials, so it is omitted.

overall Rényi efficiency is lower than the baseline. However, the efficiency difference is very slight (a -0.68% relative difference), while the corresponding drop in BLEU is still very large (-1.08 BLEU). In four cases, the best performing model in a hyperparameter setting outperformed the BASELINE, but in all of these cases, it was when all tokens were available to be marked for decomposition. It is likely that infrequent tokens which are unimpactful were selected for decomposition, and so the model performance was not negatively impacted. When restricted to only the top 2000 most frequent tokens, all models underperform the baseline.

To demonstrate that this finding is not caused purely by the smaller vocabulary size, we also experimented with setting $k = 500$ and using a new vocab size of $|\mathcal{V}| + 500$ for each BASELINE. In these cases, we observe the same negative correlation between Rényi efficiency and BLEU, indicating that the effective vocabulary size is not the salient factor.

In every case, the RANDOM-DROP BPE models had higher Rényi efficiency than the baseline but performed worse. Therefore, this provides a counterexample to Zouhar et al. (2023a).

5.2. DUPLICATION BPE

Again, we vary the hyperparameters to form different DUPLICATION BPE models. In particular, we set N (the top frequent tokens to duplicate) to 100 or 500 and k (the duplication factor) to 3 or 5. We train each configuration with three seeds and report the averages (see Table 2).[5]

In all cases, the DUPLICATION BPE tokenizer improves Rényi efficiency over the baseline and dramatically reduces BLEU (by at least 1 BLEU). The largest drops in BLEU — and simultaneously the largest gains in efficiency — are with a larger duplication factor: $k = 5$. In the pessimal case (4k/4k, $N = 500$, $k = 5$), the model was unable to converge (across 10 trials, none had higher than 5 BLEU). We conjecture that this is due to the large percentage of tokens ($1/8$ of the total vocabulary size) being duplicated 5 times, so the model is essentially forced to spread the marginal probability of a subword sequence over $|S|^5$ identical-surface-

---

[5]We do not report a **Best** for DUPLICATION BPE, as the choice of N and k represents a *specific* tokenizer. For RANDOM-DROP BPE, the choice of N and k represent a *family* of tokenizers (as $\mathcal{D}$ is selected at random each time, based on those hyperparameters).

form sequences (where $|S|$ is the sequence length). In other words, while the duplicated tokens combined might have the largest probability, this probability is divided by 5 which prevents the beam search from selecting any of the correct tokens.

In all cases, we increased the Rényi efficiency while decreasing BLEU, providing yet another counterexample to Zouhar et al. (2023a).

5.3. Other Metrics

Zouhar et al. (2023a) considered several other intrinsic metrics, introduced in Section 3, before determining that Rényi efficiency was the best correlated with downstream performance. In Table 3, we list all of these metrics to show which properly correlate with the downstream performance.

For RANDOM-DROP BPE, in all cases both **PCT** and **SEQ** correctly predict that the BPE variant will be worse than the baseline. This means that **PCT** is always lower than the baseline, **SEQ** is always higher than the baseline, and BLEU is always worse. Thus, these non-entropy-based predictors succeed where Rényi efficiency fails.

On the other hand, both **PCT** and **SEQ** fail to accurately predict the performance of DUPLICATION BPE models. For **SEQ**, the DUPLICATION BPE models always have the same value as the baseline. This is because these models do not change the surface form of the tokenization, and so, the sequence lengths are always the same. For **PCT**, the duplication of the most frequent tokens has the effect of replacing the lowest frequency tokens in the percentile range, thus raising the overall probability mass of the range and increasing the metric. Since all DUPLICATION BPE models have lower BLEU than the baseline, these metrics fail similarly to Rényi efficiency on this family.

6. Conclusion

We proposed two families of BPE modifications which break the Rényi efficiency hypothesis by Zouhar et al. (2023a). We were able to construct tokenizers such that the Rényi efficiency *negatively* correlates with the downstream performance. Our tokenizer families are designed specifically to increase Rényi efficiency over their baseline tokenizers, and, across all experiments, produce models that have much lower performance on a German→English translation task. That our results hold across a range of hyperparameter settings indicates that our tokenizer families serve as counterexamples to the Rényi efficiency hypothesis. We also find that the other metrics by Zouhar et al. (2023a) fail to correctly predict the performance of our synthetic tokenizer families.

Tokenizer	N	k	PCT ↑	SEQ ↓	BLEU
BASE. (4k/4k)	-	-	0.461	25.50	33.74
RANDOM-DROP	2k	500	0.356	31.46	33.39
	2k	1k	0.233	40.37	32.76
	4k	500	0.405	29.23	33.72
	4k	1k	0.352	33.37	33.40
RANDOM-DROP (4.5k/4.5k)	2k	500	0.356	31.46	33.35
	4.5k	500	0.402	27.93	33.69
DUPLICATE	100	3	0.590	25.50	32.37
	100	5	0.633	25.50	31.32
	500	3	0.571	25.50	32.26
	500	5	0.605	25.50	N/A
BASE. (6k/6k)	-	-	0.405	24.05	33.94
RANDOM-DROP	2k	500	0.312	29.46	33.46
	2k	1k	0.210	37.64	32.86
	6k	500	0.383	25.79	33.69
	6k	1k	0.354	27.92	33.60
RANDOM-DROP (6.5k/6.5k)	2k	500	0.303	29.19	33.37
	6.5k	500	0.376	25.35	33.69
DUPLICATE	100	3	0.525	24.05	32.27
	100	5	0.574	24.05	31.60
	500	3	0.522	24.05	32.43
	500	5	0.562	24.05	30.57
BASE. (9k/9k)	-	-	0.356	22.91	33.60
RANDOM-DROP	2k	500	0.275	28.04	33.35
	2k	1k	0.190	35.56	32.73
	9k	500	0.347	23.81	33.68
	9k	1k	0.332	25.07	33.59
RANDOM-DROP (9.5k/9.5k)	2k	500	0.273	27.70	33.31
	9.5k	500	0.339	23.83	33.60
DUPLICATE	100	3	0.456	22.91	32.48
	100	5	0.507	22.91	31.64
	500	3	0.469	22.91	32.42
	500	5	0.514	22.91	30.93
BASE. (14k/14k)	-	-	0.308	21.95	33.59
RANDOM-DROP	2k	500	0.244	26.61	33.05
	2k	1k	0.178	33.03	32.63
	14k	500	0.301	22.67	33.41
	14k	1k	0.295	23.33	33.35
RANDOM-DROP (14.5k/14.5k)	2k	500	0.241	26.50	33.11
	14.5k	500	0.301	22.43	33.49
DUPLICATE	100	3	0.377	21.95	32.49
	100	5	0.432	21.95	31.45
	500	3	0.411	21.95	32.55
	500	5	0.458	21.95	30.70

Table 3: The Percentile Frequency (**PCT**) and Sequence Length (**SEQ**) predictors compared with the BLEU score. **PCT** positively correlates with BLEU, while **SEQ** correlates negatively.

Application. Despite our work showing counterexamples of where Rényi efficiency fails, it should not be interpreted as a discouragement to use this metric for estimation of tokenization quality. We show that there exist synthetic counterexamples in the space of *all* possible tokenizations. However, a practitioner might be deciding between two real tokenizers, based on BPE variants and hyperparameters. Our counterexample lie outside of the space of commonly used tokenizers.

Related Work. In our setup, we assumed a strict separation between the tokenization and model training step. However, this need not be the case and (Xiao et al., 2010; Hiraoka et al., 2021) successfully merge these two steps into one, creating a single pipeline which jointly optimizes the tokenizer and model for the downstream task. There are also plethora mechanisms by which tokenization problems can be circumvented during training or inference, such as marginalization over possible tokenizations (Kudo, 2018; He et al., 2020; Provilkov et al., 2020). These are not currently taken into consideration and their interaction with the hypotheses surrounding predicting NLP model performance based on the tokenization is unknown. We further acknowledge tokenization-less NLP models which do not necessitate a tokenization metric, though as of 2023 they remain a minority.

Future Work. We identified a variance dimension which contradicts the Rényi efficiency hypothesis. However, the Rényi efficiency tokenization hypothesis itself was created to replace the Rényi entropy tokenization hypothesis because the latter did not take the dimension of vocabulary size into account. In the same vein, we hope the original hypothesis can be further refined to incorporate the variance from our two counterexamples.

Acknowledgements

Bibliographical References

Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. 2019. How much does tokenization affect neural machine translation? In *International Conference on Computational Linguistics and Intelligent Text Processing*, pages 545–554. Springer.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2021. Joint optimization of tokenization and downstream model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 244–255, Online. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Xinyan Xiao, Yang Liu, Young-Sook Hwang, Qun Liu, and Shouxun Lin. 2010. Joint tokenization and translation. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1200–1208, Beijing, China. Coling 2010 Organizing Committee.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023a. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. 2023b. A formal perspective on byte-pair

encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada. Association for Computational Linguistics.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. 2023c. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada. Association for Computational Linguistics.

A. BPE Training

Algorithm 2 gives pseudocode for how a BPE vocabulary is constructed. As noted in (Zouhar et al., 2023c), this formulation overcounts repeated characters — e.g., a a a, which would count $[a, a]$ twice, even though that sequence can only form one aa merge. However, we use subword-nmt (Sennrich et al., 2016) as our baseline BPE implementation, which also includes this overcounting.

Algorithm 2: Standard BPE Training
Inputs: Corpus $\mathcal{C}$, Character-set Σ, # merges n,
Output: Vocabulary $\mathcal{V}$, Merges μ

1: $\mathcal{V} \leftarrow \Sigma$
2: **for** $i \in 1 \ldots n$ **do**
3:     $(x, y) \leftarrow \underset{[a,b] \in \Sigma \times \Sigma}{\arg\max} \text{COUNT}([a, b], \mathcal{C})$
4:     APPEND$(\mathcal{V}, xy)$
5:     APPEND$(\mu, (x, y))$
6:     $\mathcal{C} \leftarrow \text{APPLY}([x, y], \mathcal{C})$
7: **end for**
8: **return** $\mathcal{V}, \mu$