THE FOUNDATIONS OF TOKENIZATION: STATISTICAL AND COMPUTATIONAL CONCERNS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tokenization—the practice of converting strings of characters from an alphabet into sequences of tokens over a vocabulary—is a critical step in the NLP pipeline. The use of token representations is widely credited with increased model performance but is also the source of many undesirable behaviors, such as spurious ambiguity or inconsistency. Despite its recognized importance as a standard representation method in NLP, the theoretical underpinnings of tokenization are not yet fully understood. In particular, the impact of tokenization on statistical estimation has been investigated mostly through empirical means. The present paper contributes to addressing this theoretical gap by proposing a unified formal framework for representing and analyzing tokenizer models. Based on the category of stochastic maps, this framework enables us to establish general conditions for a principled use of tokenizers, and most importantly, the necessary and sufficient conditions for a tokenizer model to preserve the consistency of statistical estimators. Additionally, we discuss statistical and computational concerns crucial for designing and implementing tokenizer models, such as inconsistency, ambiguity, tractability, and boundedness. The framework and results advanced in this paper contribute to building robust theoretical foundations for representations in neural language modeling that can inform future empirical research.

1 INTRODUCTION

As a critical step in the natural language processing (NLP) pipeline, tokenization generally refers to the process of breaking up sequences of symbols into subsequences that can be represented as units or "tokens". The tokenization of linguistic data has long been a common practice in the processing of natural language (cf. Palmer, 2000; Jurafsky & Martin, 2024). However, the significance of tokenizers took a turn with the emergence of deep neural models for NLP, where the representation of linguistic units plays a renewed fundamental role. With the development and widespread adoption of the *Byte Pair Encoding* (BPE) algorithm (Sennrich et al., 2016), subword tokenization became the privileged representation method for neural NLP. Adapting an existing compression algorithm (Gage, 1994) to overcome the challenges of out-of-vocabulary (OOV) terms in the context of neural machine translation, BPE quickly replaced previous heuristic and rule-based tokenizer models such as Morfessor (Creutz & Lagus, 2002) and Moses (Koehn et al., 2007), and was soon followed by other data-driven models, including *WordPiece* (Wu et al., 2016, following Schuster & Nakajima, 2012) and *Unigram* (Kudo, 2018) among the most widely adopted (cf. Mielke et al., 2021, for a survey).

The importance of subword tokenization for language models (LMs) has grown ever since and are now built into standard language modeling toolkits, remaining the only major step not fully integrated into widely used end-to-end neural models. Among their recognized benefits, two are often advanced in the literature. Tokenizers offer the capacity of training language models over an *open vocabulary*, circumventing the difficulties associated with OOV terms (Sennrich et al., 2016). Also, tokenization is often described as an efficient, lossless *encoding* of the original data (Zouhar et al., 2023a). Moreover, based on empirical evidence of different kinds, tokenization has been hypothesized to introduce a helpful inductive bias in language modeling (Nawrot et al., 2023; Schmidt et al., 2024; Uzan et al., 2024), although in the current state of the art, this hypothesis remains an open question. At the same time, tokenizers have also been in the spotlight for exhibiting undesirable behaviors that can have a negative impact on LMs. To name just a few, tokenization can be the source of spurious ambiguity (Kudo, 2018; Cao & Rimell, 2021), generate alignment issues (Poesia et al., 2022; Athiwaratkun

et al., 2024), hinder robustness (Kudo, 2018; Xue et al., 2022), neglect relevant linguistic features (Bostrom & Durrett, 2020; Hofmann et al., 2021; Gow-Smith et al., 2022; Beinborn & Pinter, 2023) or result in inconsistent scoring in the use of LMs in other scientific fields, like psycholinguistics (Salazar et al., 2020; Kauf & Ivanova, 2023).

The uncovering of undesirable behaviors together with the lack of conclusive theoretical explanations for either their positive or negative effects in language modeling motivated several recent attempts to dispense with tokenization altogether (Xue et al., 2022; Clark et al., 2022; Wang et al., 2024, *inter alia*). However, in the current state of research, the practical benefits of token representations in neural language modeling seem to outweigh their disadvantages, indicating that there is something to be understood rather than discarded in the process of tokenization.

The study of tokenization models has been an active area of research in recent years. Most of the work in this direction has been driven by an empirical perspective (Ding et al., 2019; Hou et al., 2023; Domingo et al., 2023; Fujii et al., 2023, *inter alia*). Among the notable exceptions adopting a more theoretical approach are Guo (1997); Kudo (2018); Zouhar et al. (2023b;a); Berglund & van der Merwe (2023); Rajaraman et al. (2024). However, in the current state of the art, we still see a need for a more foundational perspective from which *necessary conditions for the consistent use of tokenizers* can be formally established *in general*. Such a perspective should also provide the means to analyze known issues in tokenization in a formal way, ultimately informing future empirical research and contributing to increasing the reliance on models in situations in which properties such as formal guarantees, verification, or interpretability are as important as performance for an LM.

Accordingly, the objective of the present paper is to take a decisive step forward toward a robust theoretical grounding for neural NLP by laying the foundations of tokenization from a formal perspective. To that end, we characterize the problem of tokenization in current language modeling as arising from the fact that, in practice, starting from an alphabet Σ of elementary units, one seeks to estimate a probability distribution over Σ^* indirectly, that is, by estimating a probability distribution over sequences of tokens in Δ^*, where the set of tokens Δ is, in general, different from Σ. Therefore, the problem of tokenization is determined by the forward and backward mappings between Σ^* and Δ^*. To address this problem, we propose a formal framework based on what we found to be the simplest mathematical tool allowing us to characterize tokenizer models in their full generality, namely the category of stochastic maps. The proposed framework enables us to establish general conditions for a principled use of tokenizers. Crucially, we prove the necessary and sufficient conditions for a tokenizer model to preserve the consistency of statistical estimators. Additionally, this paper aims to advance the theoretical understanding of existing challenges associated with tokenization, particularly those pertaining to inconsistency, ambiguity, tractability, and boundedness. To achieve this, we characterize these known issues through the lens of formal properties of composable maps, such as injectivity, multiplicativity, and finite decomposition.

The plan of the paper is as follows. In §2, we present preliminary notions, including elementary aspects of formal language theory and the concept of stochastic maps, extending some existing results to cover the case of countably infinite sets. We also provide notational and terminological remarks. In §3, we propose a unified formal framework for representing and analyzing tokenization models and establish various results for the use of tokenizers, including the necessary and sufficient conditions for a tokenizer model to preserve the consistency of estimators. Finally, in §§ 4 and 5 we discuss, from a formal perspective, statistical and computational concerns relevant to the study, design, and implementation of tokenizer models.

2 Preliminaries

2.1 Formal Languages, Estimators, and Stochastic Maps

An **alphabet** Σ is a finite, non-empty set of **symbols**. The set $\Sigma^n \stackrel{\text{def}}{=} \overbrace{\Sigma \times \cdots \times \Sigma}^{n \text{ times}}$ consists of **strings** of symbols of length n. The symbol ε denotes the empty string of length 0. The union $\Sigma^* \stackrel{\text{def}}{=} \bigcup_{n=0}^{\infty} \Sigma^n$ consists of all finite strings (including ε) from the alphabet Σ. Similarly, we denote by $\Sigma^{\leq N}$ the set of all strings from Σ of length less or equal to N. String concatenation is an

associative product $\Sigma^* \times \Sigma^* \xrightarrow{\cdot} \Sigma^*$ for which ε is an identity element. The triple $(\Sigma^*, \cdot, \varepsilon)$ defines a **monoid**, which is, in fact, a model of the free monoid on the set Σ.

A **language** L over an alphabet Σ is a set of strings $L \subseteq \Sigma^*$. A **language model** p is a probability distribution over Σ^*. That is, p is a function $p : \Sigma^* \to [0, 1]$ such that $\sum_{\boldsymbol{\sigma} \in \Sigma^*} p(\boldsymbol{\sigma}) = 1$. Language models generalize languages in the sense that the support of a language model, i.e., $\operatorname{supp}(p) = \{\boldsymbol{\sigma} \mid p(\boldsymbol{\sigma}) \neq 0\}$, is a language. The definition of a language model as a probability distribution on Σ^* is deliberately broad. In particular, note that no compatibility between p and the monoidal structure in Σ^* is assumed.

In NLP, practitioners generally seek to **estimate** a language model p from exemplars of naturally occurring text. Formally, the modeler assumes there exists a true distribution $p^\star$ over Σ^*, and considers a multiset of naturally occurring texts $\{\boldsymbol{\sigma}_n\}_{n=1}^N \subset \Sigma^*$ to be samples from $p^\star$. In its most general form, an estimator of $p^\star$ is a sequence $\{p_n\}$ of probability distributions on Σ^* such that p_n becomes closer to $p^\star$ as n increases. We call an estimator **consistent** if the sequence $\{p_n\}$ converges *pointwise* to $p^\star$.[1] More precisely, given a probability distribution $p^\star \colon \Sigma^* \to [0, 1]$, and a sequence of distributions $\{p_n \colon \Sigma^* \to [0, 1]\}$, we say that $\{p_n\}$ is a consistent estimator of $p^\star$ if and only if, for all strings $\boldsymbol{\sigma} \in \Sigma^*$, the sequence of numbers $\{p_n(\boldsymbol{\sigma})\}$ converges to the number $p^\star(\boldsymbol{\sigma})$.

This notion of consistent estimation is general enough to include many estimation methods, where the p_i can depend on various properties of the sample, such as the size N, or possibly a set of parameters θ. Likewise, we use pointwise convergence to define consistent estimation because pointwise convergence is a weak kind of convergence, and so our definition is compatible with a wide variety of convergence measures, and relatively easy to check for. For example, a common practice in NLP is to produce an estimator $\{p_n\}$ through a sequence of steps in the process of minimizing *cross entropy loss*, which amounts to minimizing the *relative entropy* also called *the Kullback–Leibler divergence*, $D_{\mathrm{KL}}(p^\star \parallel p_n)$ between $p^\star$ and p_n. This is a stronger form of convergence: if $D_{\mathrm{KL}}(p^\star \parallel p_n) \to 0$ then $p_n \to p^\star$ pointwise (a consequence of Pinsker's lemma) and so $\{p_n\}$ is a consistent estimator of $p^\star$.

Our definition of tokenizer models will require the use of a special kind of map between sets called a stochastic map. The reference Baez & Fritz (2014) contains a detailed introduction to the category of finite sets with stochastic maps between them. Here, we will extend some of the results in Baez & Fritz (2014) to cover the case of countably infinite sets. We assume all sets are countable, either finite or countably infinite. A **stochastic map** from a set X to a set Y is a function from X to the set of probability distributions on Y. We use

$$X \rightsquigarrow Y$$

to denote a stochastic map from X to Y and the notation $x \mapsto f(y \mid x)$ to denote the probability of $y \in Y$ in the distribution assigned to $x \in X$. In other words, a stochastic map $f \colon X \rightsquigarrow Y$ is a function

$$\begin{aligned} X \times Y &\to [0, 1] \\ (x, y) &\mapsto f(y \mid x) \end{aligned}$$

satisfying $\sum_{y \in Y} f(y \mid x) = 1$ for all $x \in X$. The notation $f(y \mid x)$ is evocative of the conditional probability of y given x, but it's more accurate to think of "indexing" by x rather than "conditioning" on x since there is no assumption that the numbers $f(y \mid x)$ assemble into a joint distribution on $X \times Y$. In particular, the sum $\sum_{x \in X} f(y \mid x)$ could, for example, be infinite.

Significantly, stochastic maps can be composed. The composition

$$\begin{tikzcd} X \arrow[r, squiggly, "f"] \arrow[rr, squiggly, bend right, "gf"'] & Y \arrow[r, squiggly, "g"] & Z \end{tikzcd}$$

$gf \colon X \rightsquigarrow Z$ is defined by

$$gf(z \mid x) = \sum_{y \in Y} g(z \mid y) f(y \mid x). \tag{1}$$

[1] Following common convention, we will sometimes denote convergence as $\{p_n\} \to p^\star$, which is not to be confused with the notation for functional types (e.g., $\Sigma^* \to \Delta^*$). The context of use should be enough to prevent any ambiguity.

Since the sum in equation 1 is infinite, it requires a check that the formula for $gf(z \mid x)$ is finite, and that for each $x \in X$, $gf(- \mid x)$ defines a probability distribution on Z, both of which follow from the fact that:

$$\sum_{z \in Z} gf(z \mid x) = \sum_{z \in Z} \sum_{y \in Y} g(z \mid y) f(y \mid x) = \sum_{y \in Y} \sum_{z \in Z} g(z \mid y) f(y \mid x) = \sum_{y \in Y} f(y \mid x) = 1.$$

If one arranges a stochastic map into an $|X| \times |Y|$ matrix with the $f(y \mid x)$ entry in the x, y position, then every entry is nonnegative and the sum of every row is 1. The computation above shows that composition of stochastic maps is realized by matrix multiplication, and that even when the matrices are infinite, the row-column dot products are finite and the result of matrix multiplication is again a matrix with nonnegative entries whose rows sum to 1. From this perspective, it is clear that composition of stochastic maps is associative.

Stochastic maps generalize both ordinary probability distributions and functions. A probability distribution over a set X can be represented as a stochastic map into X from a 1-element set, denoted as $\mathbf{1} := \{1\}$, i.e., $p \colon \mathbf{1} \rightsquigarrow X$. In such cases, the customary notation $p(x)$ can be used without risk of ambiguity as a shorthand of the more cumbersome $p(x \mid 1)$. An ordinary function $f \colon X \to Y$ can be regarded as a stochastic map $X \rightsquigarrow Y$ by mapping x to the probability distribution on Y concentrated on the singleton $\{f(x)\}$, in which case we say the stochastic map f is *deterministic*. For simplicity, when a stochastic map $f : X \rightsquigarrow Y$ is deterministic, writing $y = f(x)$ means that $f(y \mid x) = 1$ and $f(y' \mid x) = 0$ for $y' \neq y$. Composition generalizes both composition of functions and the pushforward of a probability function via a function. If $p \colon \mathbf{1} \rightsquigarrow X$ is a probability distribution on X and $f : X \to Y$ is a deterministic function, then the composition $\mathbf{1} \overset{p}{\rightsquigarrow} X \overset{f}{\rightsquigarrow} Y$ is a stochastic map $fp : 1 \rightsquigarrow Y$, which is a probability distribution on Y whose formula is $fp(y) = \sum_{x \in X} f(y|x)p(x|1) = \sum_{x \in f^{-1}(y)} p(x)$. That is, fp is just the pushforward of the probability distribution p via the function f.

For any set X, the identity function on X behaves as an identity for stochastic maps. That is $\mathrm{id}_X \colon X \rightsquigarrow X$ is the stochastic map defined by $\mathrm{id}_X(x' \mid x) = 1$ when $x' = x$ and $\mathrm{id}_X(x' \mid x) = 0$ when $x' \neq x$. In matrix representation, id_X is the identity matrix, and satisfies $f\mathrm{id}_X = f = \mathrm{id}_Y f$ for all stochastic maps $f : X \rightsquigarrow Y$.

The notions of injectivity and surjectivity exist for stochastic maps. A stochastic map $f : X \rightsquigarrow Y$ is **injective** iff the support of $f(- \mid x)$ and $f(- \mid x')$ are disjoint whenever $x \neq x'$. The **support** of the stochastic map f is the union of the support of the distributions $f(- \mid x)$ as x ranges over X. A stochastic map $f : X \rightsquigarrow Y$ is **surjective** iff, for all $y \in Y$, there exists $x \in X$ such that $f(y \mid x) \neq 0$. Injectivity and surjectivity for stochastic maps reduce to their ordinary definitions for deterministic functions.

2.2 Notation and Terminology

We adopt the following notational conventions. Alphabets will be denoted by uppercase Greek letters (e.g., Σ, Δ). In the context of tokenization, we will be interested in looking at maps between strings of languages over two different alphabets, which we will denote as Σ and Δ. For a more intuitive presentation that avoids ambiguity, we reserve the term **alphabet** for the former and call **vocabulary** the latter instead. We denote symbols by lowercase Greek letters, e.g., $\sigma \in \Sigma, \delta \in \Delta$, calling them **characters** in the first case and **tokens** in the second. Strings will be denoted by bold lowercase Greek letters, e.g., $\boldsymbol{\sigma} \in \Sigma^*, \boldsymbol{\delta} \in \Delta^*$, reserving the name character strings or **texts** for the former and token strings or **token sequences** for the latter. The reader should keep in mind these terminological distinctions are for expository purposes only. From the formal perspective advanced in this paper, we do not assume any inherent privilege of Σ over Δ, focusing, instead, on how their respective elements can be mapped into each other.

When necessary, we will distinguish the empty character string $\varepsilon_\Sigma \in \Sigma^*$ from the empty token sequence $\varepsilon_\Delta \in \Delta^*$. Examples of strings and tokens will be written in monospace font (e.g., `t`, `the`). There are cases where $\Delta \cap \Sigma^* \neq \emptyset$, and it will be necessary to distinguish between concatenation in Σ^* and Δ^*. In Δ^*, concatenation will be denoted as $\shortmid$. So, for example, if $\Sigma = \{\mathtt{t}, \mathtt{h}, \mathtt{e}\}$ and $\Delta = \{\mathtt{th}, \mathtt{he}, \mathtt{e}\}$, the expression $\mathtt{t} \cdot \mathtt{h} \cdot \mathtt{e}$ denotes the concatenation in Σ^* of the three characters `t`, `h`, and `e`, while the expression $\mathtt{t} \shortmid \mathtt{he}$ represents the concatenation in Δ^* of the two tokens `t` and

he. The cases when $\Delta \cap \Sigma^* \neq \emptyset$ are of sufficient significance that we shall generally avoid using simple juxtaposition of characters to express concatenation. Therefore, the reader should always interpret th as a token in Δ, and not a text in Σ^* (written $\mathtt{t} \cdot \mathtt{h}$). If further notational clarification is needed, square brackets may be used to represent the concatenation of two texts in Σ^* (and likewise for Δ^*). For instance, $[\mathtt{t} \cdot \mathtt{h}] \cdot \mathtt{e}$ denotes the concatenation of the text $\mathtt{t} \cdot \mathtt{h}$ with the character e in Σ^*. Should any ambiguity between specific characters and tokens arise (e.g., $\mathtt{t} \in \Sigma$ vs. $\mathtt{t} \in \Delta$), it will be explicitly disambiguated whenever there is a risk that context alone is insufficient.

3 A Formal Framework for Tokenization

As observed in the previous pages, in current NLP, the problem of tokenization arises from the fact that one seeks to estimate a model $p^\star$ over strings of symbols in one alphabet *indirectly*, that is, by estimating a probability distribution q over strings of symbols on a different alphabet. Therefore, from a strictly formal perspective, the problem of tokenization can be characterized as that of the respective mappings between two sets of strings, conventionally referred to as the set Σ^* of character strings and the set Δ^* of token sequences. In order to estimate $p^\star$ through q, Σ^* needs to be mapped into and from Δ^*. The connection between Σ^* and Δ^* is thus made through a pair of mappings (τ, κ) that constitutes the basis of our formal characterization of tokenization. Accordingly, in its most general form, a tokenizer can be defined as follows:

Definition 3.1. *A **tokenizer model** (or simply **tokenizer**) from Σ^* to Δ^* is a pair of stochastic maps $\mathcal{T} = (\tau, \kappa)$, respectively called the **encoder** and the **decoder**, where the encoder is a stochastic map $\tau \colon \Sigma^* \rightsquigarrow \Delta^*$, and the decoder is a stochastic map $\kappa \colon \Delta^* \rightsquigarrow \Sigma^*$.*

Definition 3.1 is deliberately broad, covering *any* pair of string-to-string mappings τ and κ. Other than the fact that the domain of each mapping constitutes the codomain of the other, we define the encoder and decoder as arbitrary stochastic maps. In other words, we will be regarding τ and κ primarily from the point of view of their *composition*. In particular, we do not require any specific connection between the alphabet Σ and the vocabulary Δ, and hence the use of the terms encoder and decoder is also strictly conventional. However, the distinction is motivated by an implicit assumption behind the established use of tokenizers in language models—namely, that the samples $\{\boldsymbol{\sigma}_n\}_{n=1}^N \subset \Sigma^*$ of naturally occurring texts used for estimation can be mapped into Δ^* in such a way that the estimated model q can be, in turn, transformed into a model p over Σ^* through the map κ, such that $\kappa q = p$ can be considered as an estimate of the original distribution $p^\star$.

Despite the potential empirical increase in a model's predictive performance resulting for specific tokenization choices, the soundness of such a procedure is not guaranteed for arbitrary τ and κ without further conditions. On one hand, the notion of estimation in Δ^* is not well-defined unless there exists a reference distribution $q^\star$ over Δ^* to which the estimator $\{q_n\}$ can converge. On the other, assuming such an estimator is consistent, transforming it into a consistent estimator of $p^\star$ requires a way to map the sequence $\{q_n\}$ into a sequence $\{p_n\}$ that converges to $p^\star$.

Assuming a reference distribution $p^\star$ exists on Σ^*, one obtains a reference $q^\star$ on Δ^* simply as the composition (Eq. (1)) with the encoder: $q^\star = \tau p^\star$. In other words, the following diagram of stochastic maps commutes

$$\begin{tikzcd} & \mathbf{1} \arrow[dl, squiggly, "p^\star"'] \arrow[dr, squiggly, "q^\star"] & \\ \Sigma^* \arrow[rr, squiggly, "\tau"'] & & \Delta^* \end{tikzcd}$$

The distribution $q^\star$ is just the **pushforward** of the measure $p^\star$ along τ, which then makes the encoder τ a *measure-preserving map* between $(\Sigma^*, p^\star)$ and $(\Delta^*, q^\star)$

In the same way, $\{p_n\}$ can be obtained by mapping the sequence $\{q_n\}$ through κ. By defining $p_i = \kappa q_i$, we obtain the following commutative diagram

$$\begin{tikzcd} & \mathbb{N} \arrow[dl, squiggly, "\{q_n\}"'] \arrow[dr, squiggly, "\{p_n\}"] & \\ \Delta^* \arrow[rr, squiggly, "\kappa"'] & & \Sigma^* \end{tikzcd}$$

So far, none of these requirements imposes conditions on τ and κ other than being well-defined mappings between their respective domains and codomains. Notably, the notion of estimation of $\tau p^\star$ is well defined for arbitrary τ. However, given a consistent estimator $\{q_n\}$ of $q^\star$, $\{\kappa q_n\}$ *is not guaranteed to converge to* $p^\star$ without further conditions on κ. To establish such conditions, we will need the following lemmas.[2]

Lemma 3.1. *Let $\{p_n\}$ be a sequence of probability distributions over a countable set X that converges pointwise to a probability distribution p. Then* $\lim_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| = 0$.

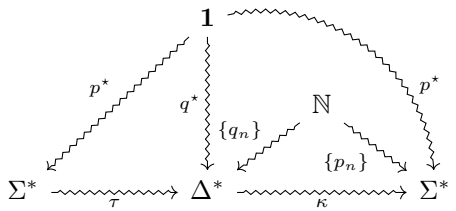
Corollary 3.0.1. *Let $\{p_n\}$ be a sequence of probability distributions over a countable set X that converges pointwise to a probability distribution p. Then $\{p_n\} \to p$ uniformly.*

Lemma 3.2. *Let f be a stochastic map from X to Y, and $\{p_n\}$ be an estimator for a probability distribution p on X. Then fp_n is an estimator for the probability distribution fp on Y.*

In other words, Lemma 3.2 says that *stochastic maps preserve the consistency of estimators*. Armed with this lemma, it is now easy to establish a simple but **fundamental principle** for the use of tokenization models in language modeling:

Theorem 3.1 (Fundamental Principle of Tokenization)**.** *Given a reference probability distribution $p^\star$ over Σ^*, a tokenizer $\mathcal{T} = (\tau, \kappa)$ from Σ^* to Δ^*, and a consistent estimator $\{q_n\}$ of the image reference distribution $q^\star = \tau p^\star$, the sequence $\{\kappa q_n\}$ is a consistent estimator of $p^\star$ if and only if $\kappa\tau p^\star = p^\star$.*

Our whole setting can be represented with the following diagram:



This setting is quite general, and Theorem 3.1 characterizes precisely when a consistent estimator $\{q_n\}$ of $q^\star$ yields a consistent estimator $\{p_n\}$ of $p^\star$ after decoding. Based on the fundamental principle expressed in Theorem 3.1, we propose the following definitions:

Definition 3.2. *Given a probability distribution p over Σ^*, a tokenizer $\mathcal{T} = (\tau, \kappa)$ from Σ^* to Δ^* is* ***consistent with respect to*** *p if we have $\kappa\tau p = p$.*

Definition 3.3. *Let p be a probability distribution over Σ^* and $\mathcal{T} = (\tau, \kappa)$ a tokenizer from Σ^* to Δ^*. When $\kappa\tau = \mathrm{id}_{\Sigma^*}$, we say that $\mathcal{T}$ is* ***exact****.*

Notice that exact tokenizers are consistent, but a tokenizer that is consistent with respect to a distribution p is not necessarily exact. Take, for instance, a probability distribution p over some set X and $x', x'' \in X$ such that $p(x') = p(x'') = c$. Then one can fashion a tokenizer for which $\kappa\tau(x) = x$ for all x except $\kappa\tau(x') = x''$ and $\kappa\tau(x'') = x'$. Such a tokenizer is consistent with respect to p without being exact. Consistency with respect to all distributions, however, is the same as being exact.

Proposition 3.1. *A tokenizer $\mathcal{T} = (\tau, \kappa)$ from Σ^* to Δ^* is exact if and only if it is consistent with respect to every probability distribution over Σ^*.*

Significantly, exact tokenizers have special properties. First, if a tokenizer (τ, κ) is exact, then κ is deterministic over the image of τ, because id_{Σ^*} also is. Formally:

Proposition 3.2. *Let $\mathcal{T} = (\tau, \kappa)$ be an exact tokenizer from Σ^* to Δ^*. Then κ is deterministic on the support of τ (or, κ is deterministic "τ almost everywhere.")*

The condition $\kappa\tau = \mathrm{id}_{\Sigma^*}$ means τ is a right inverse (or **section**) of κ and κ is a left inverse (or **retraction**) of τ. The proof of Proposition 3.2 shows that $\tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma})$ places probability zero on every $\boldsymbol{\delta} \in \Delta^*$ such that $\kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta}) = 0$. Since κ is deterministic in this case, it follows that for an exact tokenizers (τ, κ), the encoder does not place positive probability mass on a token sequence for more than one text. In other words, τ is injective. Additionally, it must be that for each text $\boldsymbol{\sigma}$ there is a $\boldsymbol{\delta}$ with $\kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta}) = 1$, and therefore κ is surjective.

[2]The proofs for all formal results (theorems, propositions, lemmas) have been placed in the Appendix.

4 Statistical Concerns: Inconsistency and Ambiguity

While in most concrete cases of statistical language modeling a tokenizer's consistency is implicitly or explicitly assumed, there are many ways in which the conditions established in the previous section can, and in practice do, fail to be satisfied. In this section, we discuss two main statistical concerns to be considered when implementing or using tokenizers, namely inconsistency and ambiguity, and associate them with the properties of maps introduced in the previous section. The following definitions will be convenient:

Definition 4.1. *Given a tokenizer $\mathcal{T} = (\tau, \kappa)$, we say $\mathcal{T}$* ***has a deterministic encoder*** *(resp.* ***decoder****) or is* ***τ-deterministic*** *(resp.* ***κ-deterministic****) if τ (resp. κ) is a deterministic map. When a tokenizer $\mathcal{T}$ is both τ-deterministic and exact, we have that $\mathcal{T}$ is also κ-deterministic, and $\kappa = \tau^{-1}$ over $\tau(\Sigma^*)$. Therefore, in such a case we say $\mathcal{T}$ is* ***strictly bijective****.*

Noninjective τ and Inconsistency. Most commonly used tokenizers have deterministic encoders, including BPE (Sennrich et al., 2016) and WordPiece (Wu et al., 2016), as well as Unigram (Kudo, 2018) when used without regularization. As we have seen, deterministic functions can be understood as a particular case of stochastic maps where the probability mass is concentrated on one element. Tokenizers with deterministic encoders thus constitute a simplified form of tokenization. However, even in this simplified setting, the consistency of the tokenization process is not guaranteed. Example 8.1 in the appendix offers an elementary intuition of this circumstance.

As shown in §3 and illustrated in Example 8.1, a fundamental cause of a tokenizer's inconsistency is the *lack of injectivity of the encoder τ*. This is not just a theoretical concern. Even if in its abstract specification a tokenizer's encoder may appear to be injective, implementation decisions often introduce noninjective behaviors. These include normalizing operations, such as lowercasing, stripping accents, removing punctuation, or uniforming whitespaces (e.g., Moi & Patry, 2023). Irrespective of how the core tokenization function is defined, including this preprocessing step as part of the tokenizer model results in a noninjective encoding that compromises the consistency of estimators.

Even if all text normalization is excluded from the decoding function, it can still happen that τ is undefined for some elements in Σ, and is, therefore, only a partial function. If the exceptions are handled by returning a unique distinguished token in Δ (e.g., an 'unknown' token `UNK`), then τ becomes noninjective, incurring the risk of inconsistency. The appeal to an `UNK` token and the difficulties associated with it have been widely studied from the perspective of OOV terms, especially in the context of NMT (e.g., Luong & Manning, 2016; Jean et al., 2015), ultimately leading to subword tokenizers as a way of providing "open vocabulary" solutions (Sennrich et al., 2016; Wu et al., 2016).[3] Formally, it is enough to inject Σ into Δ (i.e., to include the alphabet in the vocabulary) to achieve an open vocabulary, something most tokenizers do by default. However, open vocabulary solutions do not entirely remove the risk of noninjective decoding, which is not reducible to OOV problems. Some open vocabulary models, for instance, limit the size of Σ to the sample's k most frequent symbols, mapping all other symbols to an `UNK` character *in* Σ (e.g., Wu et al., 2016). Understood as a preprocessing step, this operation should not affect τ's injectivity. However, the use of copy models (e.g., Luong et al., 2015) that keep track of the original out-of-alphabet symbols to restore them in decoding, violates *de facto* the tokenizer's injectivity, and with it, the model's consistency over strings including those symbols.

Even if all symbols in the training sample are included in Σ, *out-of-alphabet* symbols can always be encountered at test or inference time. The recourse to a distinguished `UNK` symbol *both in* Σ *and* Δ must, therefore, be handled in such a way that the injectivity of τ is guaranteed. The use of a stochastic κ (e.g., Mielke & Eisner, 2019) over the restricted domains of out-of-alphabet/vocabulary elements could, in principle, provide novel ways of addressing this problem in agreement with consistency concerns.

Noninjective κ Ambiguity. Whenever κ is noninjective, the tokenizer introduces *ambiguity* in the model because more than one token sequence is mapped into a unique text. In strictly bijective tokenizers, decoding is injective over the encoder's image, thus preventing ambiguity in principle.

[3] For a tokenization-free alternative to the OOV problem, see, for instance, Xue et al. (2022) and Clark et al. (2022), who also offer a good overview of existing approaches to this problem.

However, in practice, whenever $\tau(\Sigma^*)$ is a proper subset of Δ^*, it may happen that the probability mass placed by the estimated language model outside the image of τ is nonzero, reintroducing ambiguity into the model (cf. Example 8.2 in the Appendix for an elementary illustration). This ambiguity is, however, *spurious* because τ was assumed to be deterministic, and hence the ambiguity does not stem from the reference distribution $p^\star$, but is a side-effect of the estimator. An obvious source of spurious ambiguity resides in the fact that consistency is a property defined *in the limit*. As a consequence, for any $\boldsymbol{\delta} \in \Delta^*$, $q_n(\boldsymbol{\delta})$ can and will generally differ from $q^\star(\boldsymbol{\delta})$. Spurious ambiguity can also result from the fact that, due to the properties of gradient descent and certain activation functions such as softmax, neural models are incapable of assigning zero probability to elements of Δ^*. While spurious ambiguity has been identified among the motivations for introducing subword regularization (Kudo, 2018; Provilkov et al., 2020), it is often overlooked or disregarded, despite its potential nonnegligible effect on estimation (Cao & Rimell, 2021), although mostly in the case of "strongly out-of-domain evaluation sets" (Chirkova et al., 2023).

Spurious ambiguity is not the only kind of ambiguity that can result from the use of tokenization in language models. Whenever a tokenizer model is stochastic, a deterministic κ must be noninjective for the model to preserve the consistency of estimators. However, the ambiguity thus introduced is not spurious in that it is deliberately designed for statistical purposes. In current tokenization practices, the main reason for the introduction of *stochastic* ambiguity is regularization (Kudo, 2018; Provilkov et al., 2020). The claim is that, by exhibiting different token sequences corresponding to the same text during training, a model increases its capability to handle text compositionality as well as its robustness to noise and tokenization errors. However, one could also conceive of a stochastic tokenizer where the possible images of a text reflect the objective probabilities of all *linguistic* ambiguities potentially affecting it (e.g., `an|icecream`, `an|ice|cream`, `a|nice|cream` as three possible token sequences for the text: `a·n·i·c·e·c·r·e·a·m`). This would require, however, to enhance tokenizers with linguistically motivated segmentation (see, for instance, Bostrom & Durrett, 2020; Hofmann et al., 2021; Gow-Smith et al., 2022; Beinborn & Pinter, 2023).

Although all these classes of ambiguity (spurious, stochastic, and linguistic) are both formally and semantically different, they all represent the same challenge for the tokenizer's consistency: The probability mass indirectly assigned by the model to one text in a language is spread out over different token sequences. Notice that all these cases of ambiguity can coexist, and hence their impact is difficult to evaluate. Yet, from a formal perspective, the solution for all these cases is the same: The computation of κq_n for a single text $\boldsymbol{\sigma} \in \Sigma^*$ requires marginalizing over all its preimages $\boldsymbol{\delta}$ through κ, for which $q_n(\boldsymbol{\delta}) > 0$, following the composition of stochastic maps presented in the previous section (Eq. (1)). However, such operation can be computationally challenging because it can imply summing over a large or even infinite number of terms. For different strategies to address that challenge, see van Merriënboer et al. (2017); Buckman & Neubig (2018); Grave et al. (2019); Hannun et al. (2020); Cao & Rimell (2021).

5 Computational Concerns: Tractability and Boundedness

As the end of the previous section shows, even when a tokenizer model is consistent and all statistical concerns are taken into account, there are still computational aspects that can hinder the practice of tokenization. In this section, we turn to issues of tractability, and boundedness.

Multiplicativity and Tractability. Definitions 3.1 to 3.3 are general enough to allow for all kinds of encoding and decoding functions, including uncomputable ones (see Example 8.3 in the Appendix for an example). However, even when a tokenizer model is computable, its **tractability** is not guaranteed. Indeed, there are many reasons that could make the computation of tokenization intractable. Many of the operations defining tokenizer models involve sums over infinite sets. This is particularly true for the composition of stochastic maps whenever it is performed over an infinite domain, as in our case. Therefore, it is crucial to assess the tractability not only of τ and κ, but also of their composition $\kappa\tau$.

We have seen that when a tokenizer model is exact, τ is a section for κ, or equivalently, τ is injective. It follows that, for any $\boldsymbol{\sigma} \in \Sigma^*$, τ concentrates the probability mass on only a subset of Δ^*. This property can help reduce the computational costs by restricting the sums to just those subsets. However, without further constraints, those subsets can still be infinite. For this reason, we consider the following properties:

Definition 5.1. *We say a tokenizer model $\mathcal{T} = (\tau, \kappa)$ is **multiplicative** if its decoder κ respects the concatenation products; that is, if $\kappa(\boldsymbol{\delta}' | \boldsymbol{\delta}'') = \kappa(\boldsymbol{\delta}') \cdot \kappa(\boldsymbol{\delta}'')$.*

Definition 5.2. *We say the **kernel** of a multiplicative tokenizer's decoder κ is **trivial** if κ maps nonempty token sequences to nonempty token sequences (i.e., if $\boldsymbol{\delta} \neq \varepsilon_\Delta$ then $\kappa(\boldsymbol{\delta}) \neq \varepsilon_\Sigma$).*

The most commonly used tokenizers, including BPE, WordPiece, and Unigram, are multiplicative. An obvious consequence of multiplicative tokenizers is that decoding preserves the prefix structure of token sequences. More precisely, let $\boldsymbol{\delta}' \preceq \boldsymbol{\delta}$ denote the fact that $\boldsymbol{\delta} = \boldsymbol{\delta}' | \boldsymbol{\delta}''$ for $\boldsymbol{\delta}, \boldsymbol{\delta}', \boldsymbol{\delta}'' \in \Delta^*$ (and likewise for $\boldsymbol{\sigma}, \boldsymbol{\sigma}', \boldsymbol{\sigma}'' \in \Sigma^*$). Then we have that $\boldsymbol{\delta}' \preceq \boldsymbol{\delta}$ implies $\kappa(\boldsymbol{\delta}') \preceq \kappa(\boldsymbol{\delta})$. This property is crucial in autoregressive models, where each token in a sequence depends only on the tokens preceding it and can then be computed in a left-to-right fashion. In these cases, multiplicativity ensures that the model's output can be decoded on the fly, excluding decoding functions such as string reversal. Moreover, notice that, for a kernel of a multiplicative tokenizer's decoder to be trivial, it is enough that $\kappa(\delta) \neq \varepsilon_\Sigma$ for any $\delta \in \Delta$. This implies that token sequences do not include special tokens that are erased during decoding, e.g., padding or end-of-sentence tokens.

Importantly, when a multiplicative tokenizer's decoder has a trivial kernel, a decoded text cannot be shorter than the token sequence from which it has been decoded. This simple observation guarantees that the number of preimages of a text under κ is finite. More precisely,

Proposition 5.1. *Let $\mathcal{T} = (\tau, \kappa)$ be a multiplicative tokenizer model whose decoder's kernel is trivial. If $\kappa(\delta_1 | \delta_2 | \ldots | \delta_m) = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_n$ then $m \leq n$.*

Corollary 5.0.1. *Let $\mathcal{T} = (\tau, \kappa)$ be a multiplicative tokenizer model whose decoder's kernel is trivial. Then for any text $\boldsymbol{\sigma}$, the set $\kappa^{-1}(\boldsymbol{\sigma})$ is finite.*

Proposition 5.1 guarantees that, if no token in the vocabulary is mapped to the empty string, then the length of every preimage of a text $\boldsymbol{\sigma}$ of length n has length less than or equal to n. So the number of elements in $\kappa^{-1}(\boldsymbol{\sigma})$, and hence also the support of τ for any given $\boldsymbol{\sigma} \in \Sigma^*$ when (τ, κ) is exact, is bounded by $\sum_{i=1}^{n} |\Delta|^i$.

Since this bound is exponential in n, the exact or approximate computation of a tokenizer's encoding and decoding so that the consistency of the language model is not compromised requires the appeal to multiple strategies as the ones mentioned at the end of §4. Ideally, the complexity of a tokenizer model (that is, of the composition $\kappa\tau$) should be at most linear in the length of the input text. By placing all the probability mass on one token sequence, $\kappa\tau$ in exact deterministic tokenizers, such as BPE and WordPiece, can be computed in linear time as long as τ and κ can be computed in linear time. However, rigorously handling spurious ambiguity still represents a challenge in these cases.

Finite Decomposition and Boundedness. Finally, even though multiplicativity ensures the length of all the preimages of $\kappa^{-1}(\boldsymbol{\sigma})$ is bounded by the length of $\boldsymbol{\sigma}$, texts themselves may have unbounded length. In practice, the bounded character of tokenization is secured externally, namely by fixing a hyperparameter that artificially limits the length of input texts. However, it can be desirable to address boundedness as an internal property of a tokenizer. For this reason, we introduce the following definitions:

Definition 5.3. *Given two sets A, B, and a map $f\colon A^* \to B^*$ we say f is of **finite type** if, there exists $n \in \mathbb{N}$ such that, for every $\boldsymbol{a} \in A^*$, there exists a nonempty $\boldsymbol{a}' \in A^{\leq n}$ such that $\boldsymbol{a} = \boldsymbol{a}' \cdot \boldsymbol{a}''$ and $f(\boldsymbol{a}) = f(\boldsymbol{a}') \cdot f(\boldsymbol{a}'')$.*

Definition 5.4. *A tokenizer model $\mathcal{T} = (\tau, \kappa)$ is called **bounded** if both τ and κ are of finite type.*

If a tokenizer is bounded, it means the input can be decomposed into bounded components. The "maximal munch" approach (Reps, 1998; Palmer, 2000) adopted by WordPiece, for instance, iteratively maps the successive longest prefixes of a text to tokens in the vocabulary. WordPiece's encoder is thus bounded by the maximum length of the preimages of Δ through τ, and can therefore be implemented as a finite-state transducer (Song et al., 2021). Notice, however, that, in general, if a tokenizer is bounded in the sense of Definitions 5.3 and 5.4, that does not mean there exists a tractable algorithm to find the decomposition $\boldsymbol{a} = \boldsymbol{a}' \cdot \boldsymbol{a}''$, which could, in principle, depend on the entire $\boldsymbol{a}$. Significantly, Berglund & van der Merwe (2023) showed that, under specific conditions on the structure of the list of rules or "merges", BPE is also bounded and an algorithm can be found to compute $\tau(\boldsymbol{\sigma}') | \tau(\boldsymbol{\sigma}'') = \tau(\boldsymbol{\sigma})$ in an online fashion with finite lookahead. Moreover, Berglund

et al. (2024) proposed an algorithm for constructing deterministic finite automata representing BPE's encoder.

6 Conclusion

In this work, we have addressed the use of token representations in NLP from a foundational perspective. Relying on the category of stochastic maps as an elementary formal tool, we proposed a general definition of a tokenizer as an arbitrary pair of composable maps. The framework proposed enabled us to formally establish several properties of tokenization, and most importantly, the necessary and sufficient condition for a tokenizer to preserve the consistency of estimators. Furthermore, our approach allowed to shed new theoretical light on known issues concerning tokenization, namely inconsistency, ambiguity, tractability, and boundedness, by characterizing the latter through formal properties of composable maps such as injectivity, multiplicativity, or finite decomposition. We believe this framework will inform future empirical research and contribute to establishing and developing theoretical and practical aspects of representation learning in NLP on solid grounds, especially in cases where the reliance on a model requires to go beyond its mere performance, and take into account properties such as formal guarantees, verification, theoretical soundness, interpretability, or liability.

7 Limitations

Statistical and computational concerns are not the only concerns relevant to a foundational approach to tokenization. In particular, this paper does not address structural concerns, i.e., structural properties of the sets Σ^* and Δ^* such as their monoidal structure, the preservation of structural features through τ and κ, or the important question of the choice of Δ. Algorithmic concerns related to the effective computation of exact tokenizers have been left unaddressed, as well as theoretical concerns related to interpretability and the possible relation to linguistic segmentation. Although the perspective adopted here is purely formal, and as such, self-contained, some aspects alluded in this paper could benefit from the insights given by experimental results. These points will be the object of future work.

References

Ben Athiwaratkun, Shiqi Wang, Mingyue Shang, Yuchen Tian, Zijian Wang, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Rob Kwiatowski, Ramesh Nallapati, and Bing Xiang. Token alignment via character matching for subword completion, 2024.

John C. Baez and Tobias Fritz. A Bayesian characterization of relative entropy. *CoRR*, abs/1402.3067, 2014. URL http://arxiv.org/abs/1402.3067.

Lisa Beinborn and Yuval Pinter. Analyzing cognitive plausibility of subword tokenization. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4478–4486, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.272. URL https://aclanthology.org/2023.emnlp-main.272.

Martin Berglund and Brink van der Merwe. Formalizing BPE tokenization. In Benedek Nagy and Rudolf Freund (eds.), *Proceedings of the 13th International Workshop on, Non-Classical Models of Automata and Applications, Famagusta, North Cyprus, 18th-19th September, 2023*, volume 388 of *Electronic Proceedings in Theoretical Computer Science*, pp. 16–27. Open Publishing Association, 2023. doi: 10.4204/EPTCS.388.4.

Martin Berglund, Willeke Martens, and Brink van der Merwe. Constructing a BPE tokenization DFA. In Szilárd Zsolt Fazekas (ed.), *Implementation and Application of Automata*, pp. 66–78, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-71112-1. URL https://link.springer.com/chapter/10.1007/978-3-031-71112-1_5.

Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4617–4624, Online, November 2020. Association for Computational

Linguistics. doi: 10.18653/v1/2020.findings-emnlp.414. URL https://aclanthology.org/2020.findings-emnlp.414.

Jacob Buckman and Graham Neubig. Neural lattice language models. *Transactions of the Association for Computational Linguistics*, 6:529–541, 2018. doi: 10.1162/tacl_a_00036. URL https://aclanthology.org/Q18-1036.

Kris Cao and Laura Rimell. You should evaluate your language model on marginal likelihood over tokenisations. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2104–2114, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.161. URL https://aclanthology.org/2021.emnlp-main.161.

Nadezhda Chirkova, Germán Kruszewski, Jos Rozen, and Marc Dymetman. Should you marginalize over possible tokenizations? In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–12, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.1. URL https://aclanthology.org/2023.acl-short.1.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5.

Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pp. 21–30. Association for Computational Linguistics, July 2002. doi: 10.3115/1118647.1118650. URL https://aclanthology.org/W02-0603.

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh. A call for prudent choice of subword merge operations in neural machine translation. In Mikel Forcada, Andy Way, Barry Haddow, and Rico Sennrich (eds.), *Proceedings of Machine Translation Summit XVII: Research Track*, pp. 204–213, Dublin, Ireland, August 2019. European Association for Machine Translation. URL https://aclanthology.org/W19-6620.

Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? In Alexander Gelbukh (ed.), *Computational Linguistics and Intelligent Text Processing*, pp. 545–554, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-24337-0. URL https://link.springer.com/chapter/10.1007/978-3-031-24337-0_38.

Takuro Fujii, Koki Shibata, Atsuki Yamaguchi, Terufumi Morishita, and Yasuhiro Sogawa. How do different tokenizers perform on downstream tasks in scriptio continua languages?: A case study in Japanese. In Vishakh Padmakumar, Gisela Vallejo, and Yao Fu (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 39–49, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-srw.5. URL https://aclanthology.org/2023.acl-srw.5.

Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, feb 1994. ISSN 0898-9788. URL https://dl.acm.org/doi/abs/10.5555/177910.177914.

Edward Gow-Smith, Harish Tayyar Madabushi, Carolina Scarton, and Aline Villavicencio. Improving tokenisation by alternative treatment of spaces. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11430–11443, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.786. URL https://aclanthology.org/2022.emnlp-main.786.

Edouard Grave, Sainbayar Sukhbaatar, Piotr Bojanowski, and Armand Joulin. Training hybrid language models by marginalizing over segmentations. In Anna Korhonen, David Traum, and Lluís

Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1477–1482, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1143. URL https://aclanthology.org/P19-1143.

Jin Guo. Critical tokenization and its properties. *Computational Linguistics*, 23(4):569–596, 1997. URL https://aclanthology.org/J97-4004.

Awni Hannun, Vineel Pratap, Jacob Kahn, and Wei-Ning Hsu. Differentiable weighted finite-state transducers, 2020. URL https://arxiv.org/abs/2010.01003.

Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. Superbizarre is not superb: Derivational morphology improves BERT's interpretation of complex words. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3594–3608, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.279. URL https://aclanthology.org/2021.acl-long.279.

Jue Hou, Anisia Katinskaia, Anh-Duc Vu, and Roman Yangarber. Effects of sub-word segmentation on performance of transformer language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7413–7425, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.459. URL https://aclanthology.org/2023.emnlp-main.459.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In Chengqing Zong and Michael Strube (eds.), *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1–10, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1001. URL https://aclanthology.org/P15-1001.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Third edition draft edition, 2024. URL https://web.stanford.edu/~jurafsky/slp3/.

Carina Kauf and Anna Ivanova. A better way to do masked language model scoring. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 925–935, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.80. URL https://aclanthology.org/2023.acl-short.80.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In Sophia Ananiadou (ed.), *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pp. 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/P07-2045.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://aclanthology.org/P18-1007.

Minh-Thang Luong and Christopher D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1054–1063, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1100. URL https://aclanthology.org/P16-1100.

Thang Luong, Ilya Sutskever, Quoc Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. In Chengqing Zong and Michael Strube (eds.), *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 11–19, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1002. URL https://aclanthology.org/P15-1002.

Sabrina J. Mielke and Jason Eisner. Spell once, summon anywhere: A two-level open-vocabulary language model. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33016843. URL https://doi.org/10.1609/aaai.v33i01.33016843.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *CoRR*, abs/2112.10508, 2021. URL https://arxiv.org/abs/2112.10508.

Anthony Moi and Nicolas Patry. HuggingFace's Tokenizers, April 2023. URL https://github.com/huggingface/tokenizers.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353.

David D. Palmer. Tokenisation and sentence segmentation. In Robert Dale, Hermann Moisl, and Harold Somers (eds.), *Handbook of Natural Language Processing*, chapter 2, pp. 24–25. Marcel Dekker, 2000.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models, 2022.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-dropout: Simple and effective subword regularization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1882–1892, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.170. URL https://aclanthology.org/2020.acl-main.170.

Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. Toward a theory of tokenization in LLMs, 2024.

Thomas Reps. "Maximal-munch" tokenization in linear time. *ACM Trans. Program. Lang. Syst.*, 20(2):259–273, mar 1998. ISSN 0164-0925. doi: 10.1145/276393.276394. URL https://doi.org/10.1145/276393.276394.

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. Masked language model scoring. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2699–2712, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.240. URL https://aclanthology.org/2020.acl-main.240.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression, 2024.

Mike Schuster and Kaisuke Nakajima. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152, 2012. doi: 10.1109/ICASSP.2012.6289079.

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

Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. Fast WordPiece tokenization. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 2089–2103, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.160. URL https://aclanthology.org/2021.emnlp-main.160.

Omri Uzan, Craig W. Schmidt, Chris Tanner, and Yuval Pinter. Greed is all you need: An evaluation of tokenizer inference methods, 2024.

Bart van Merriënboer, Amartya Sanyal, Hugo Larochelle, and Yoshua Bengio. Multiscale sequence modeling with a learned dictionary, 2017. URL https://arxiv.org/abs/1707.00762.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M. Rush. MambaByte: Token-free Selective State Space Model, 2024. URL https://arxiv.org/abs/2401.13660.

Yonghui Wu, Mike Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016. URL https://api.semanticscholar.org/CorpusID:3603249.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. Tokenization and the noiseless channel. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5184–5207, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.284. URL https://aclanthology.org/2023.acl-long.284.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 598–614, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.38. URL https://aclanthology.org/2023.findings-acl.38.

8 Appendix

8.1 Proofs

Lemma 3.1. *Let $\{p_n\}$ be a sequence of probability distributions over a countable set X that converges pointwise to a probability distribution p. Then* $\lim_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| = 0$.

Proof. Fatou's lemma applied to X with the counting measure implies that for any sequence of nonnegative functions $\{f_n\}$ on X,

$$\sum_{x\in X} \liminf_{n\to\infty} f_n(x) \leq \liminf_{n\to\infty} \sum_{x\in X} f_n(x). \tag{2}$$

We'll apply this to $f_n := p_n + p - |p_n - p|$. First, note that since $\lim_{n\to\infty} p_n(x) = p(x)$, we have $\liminf_{n\to\infty} f_n(x) = p(x) + p(x) - 0 = 2p(x)$ so the left hand side of equation 2 becomes $\sum_{x\in X} 2p(x) = 2$. Therefore,

$$\begin{aligned}
2 &\leq \liminf_{n\to\infty} \sum_{x\in X} f_n(x) \\
&= \liminf_{n\to\infty} \sum_{x\in X} p_n(x) + p(x) - |p_n(x) - p(x)| \\
&= \liminf_{n\to\infty} \sum_{x\in X} p_n(x) + \sum_{x\in X} p(x) - \sum_{x\in X} |p_n(x) - p(x)| \\
&= \liminf_{n\to\infty} 1 + 1 - \sum_{x\in X} |p_n(x) - p(x)| \\
&= 2 - \limsup_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)|.
\end{aligned}$$

It follows that $\limsup_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| \leq 0$. So $\lim_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| = 0$. □

Corollary 3.0.1. *Let $\{p_n\}$ be a sequence of probability distributions over a countable set X that converges pointwise to a probability distribution p. Then $\{p_n\} \to p$ uniformly.*

Proof. Since the sum of nonnegative numbers is always greater than any particular term in the sum and $\lim_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| = 0$, we can conclude that the sequence $\{p_n\} \to p$ uniformly. □

Lemma 3.2. *Let f be a stochastic map from X to Y, and $\{p_n\}$ be an estimator for a probability distribution p on X. Then fp_n is an estimator for the probability distribution fp on Y.*

Proof. Fix $y \in Y$. We will show that $\{fp_n(y)\} \to fp(y)$. By Lemma 3.1, we have that $\lim_{n\to\infty} \sum_{x\in X} |p_n(x) - p(x)| = 0$. Therefore,

$$\begin{aligned}
\lim_{n\to\infty} |fp_n(y) - fp(y)| &= \lim_{n\to\infty} \left| \sum_x f(y \mid x) p_n(x) - \sum_x f(y \mid x) p(x) \right| && (3) \\
&\leq \lim_{n\to\infty} \sum_x f(y \mid x) \, |p_n(x) - p(x)| && (4) \\
&\leq \lim_{n\to\infty} \sum_x |p_n(x) - p(x)| && (5) \\
&= 0. && (6)
\end{aligned}$$

□

Theorem 3.1 (Fundamental Principle of Tokenization)**.** *Given a reference probability distribution $p^\star$ over Σ^*, a tokenizer $\mathcal{T} = (\tau, \kappa)$ from Σ^* to Δ^*, and a consistent estimator $\{q_n\}$ of the image reference distribution $q^\star = \tau p^\star$, the sequence $\{\kappa q_n\}$ is a consistent estimator of $p^\star$ if and only if $\kappa\tau p^\star = p^\star$.*

Proof. By hypothesis, $\{q_n\} \to q^\star$ and by definition $q^\star = \tau p^\star$. By Lemma 3.2, applying κ to both sides, we have that $\{\kappa q_n\} \to \kappa q^\star$ and so

$$\{\kappa q_n\} \to \kappa\tau p^\star.$$

Therefore, if $\kappa\tau p^\star = p^\star$ we have $\{\kappa q_n\} \to p^\star$. Conversely, if $\{\kappa q_n\} \to p^\star$ we have both $\{\kappa q_n\} \to p^\star$ and $\{\kappa q_n\} \to \kappa\tau p^\star$ and so by the uniqueness of limits, $\kappa\tau p^\star = p^\star$. □

Proposition 3.1. *A tokenizer $\mathcal{T} = (\tau, \kappa)$ from Σ^* to Δ^* is exact if and only if it is consistent with respect to every probability distribution over Σ^*.*

Proof. Exact means $\kappa\tau = \mathrm{id}_{\Sigma^*}$ hence $\kappa\tau p = p$ for every probability distribution p on Σ^*. To prove the other direction, suppose that $\kappa\tau p = p$ for every p on Σ^*. Fix an arbitrary $\boldsymbol{\sigma} \in \Sigma^*$. Let $p_{\boldsymbol{\sigma}}$ be the point mass distribution on Σ^* concentrated on $\boldsymbol{\sigma}$. So $p_{\boldsymbol{\sigma}}(\boldsymbol{\sigma}) = 1$ and $p_{\boldsymbol{\sigma}}(\boldsymbol{\sigma}') = 0$ for any $\boldsymbol{\sigma}' \neq \boldsymbol{\sigma}$. By hypothesis, $p_{\boldsymbol{\sigma}} = \kappa\tau p_{\boldsymbol{\sigma}}$. Apply to $\boldsymbol{\sigma}$ to get $1 = \kappa\tau p_{\boldsymbol{\sigma}}(\boldsymbol{\sigma})$. The right hand side, as the pushforward of $p_{\boldsymbol{\sigma}}$ via $\kappa\tau$, says $1 = p_{\boldsymbol{\sigma}}(\kappa\tau(\boldsymbol{\sigma}))$. Since $p_{\boldsymbol{\sigma}}$ takes the value 1 at only one point, it follows that the argument $\kappa\tau(\boldsymbol{\sigma}) = \boldsymbol{\sigma}$. Since $\boldsymbol{\sigma}$ was arbitrary, we conclude that $\kappa\tau = \mathrm{id}_{\Sigma^*}$, i.e., the tokenizer (τ, κ) is exact. □

Proposition 3.2. *Let $\mathcal{T} = (\tau, \kappa)$ be an exact tokenizer from Σ^* to Δ^*. Then κ is deterministic on the support of τ (or, κ is deterministic "τ almost everywhere.")*

Proof. Assume (τ, κ) is exact, i.e., $\kappa\tau = \mathrm{id}_{\Sigma^*}$, and let $\boldsymbol{\sigma} \in \Sigma^*$.

Since $\sum_{\boldsymbol{\delta} \in \Delta^*} \tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) = 1$ and $\kappa\tau(\boldsymbol{\sigma}|\boldsymbol{\sigma}) = \mathrm{id}_{\Sigma^*}(\boldsymbol{\sigma} \mid \boldsymbol{\sigma}) = 1$. We obtain:

$$
\begin{aligned}
0 &= \sum_{\boldsymbol{\delta} \in \Delta^*} \tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) - \kappa\tau(\boldsymbol{\sigma}|\boldsymbol{\sigma}) && (7)\\
&= \sum_{\boldsymbol{\delta} \in \Delta^*} \tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) - \sum_{\boldsymbol{\delta} \in \Delta^*} \kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta})\tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) && (8)\\
&= \sum_{\boldsymbol{\delta} \in \Delta^*} \tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) - \kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta})\tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) && (9)\\
&= \sum_{\boldsymbol{\delta} \in \Delta^*} \tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma})(1 - \kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta})). && (10)
\end{aligned}
$$

Since Eq. (10) is a sum of nonnegative terms that equals zero, each terms must be zero. It follows that if $\tau(\boldsymbol{\delta} \mid \boldsymbol{\sigma}) > 0$ for some $\boldsymbol{\delta}$ (e.g.: $\boldsymbol{\delta}$ is in the support of τ) then then $1 - \kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta}) = 0 \Leftrightarrow \kappa(\boldsymbol{\sigma} \mid \boldsymbol{\delta}) = 1$. From which it follows that

$$
\kappa(\boldsymbol{\sigma}' \mid \boldsymbol{\delta}) = \begin{cases} 1 & \text{if } \boldsymbol{\sigma}' = \boldsymbol{\sigma} \\ 0 & \text{if } \boldsymbol{\sigma}' \neq \boldsymbol{\sigma}. \end{cases}.
$$

□

Proposition 5.1. *Let $\mathcal{T} = (\tau, \kappa)$ be a multiplicative tokenizer model whose decoder's kernel is trivial. If $\kappa(\delta_1 \shortmid \delta_2 \shortmid \ldots \shortmid \delta_m) = \sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_n$ then $m \leq n$.*

Proof. We can reason by induction. The property is true when $m = 1$ since 1 is the minimun length of any possible image of κ. Assume it is true for $m = k$ and let $\boldsymbol{\delta} = \delta_1 \shortmid \ldots \shortmid \delta_k \shortmid \delta_{k+1}$. Then $\kappa(\boldsymbol{\delta}) = \kappa(\delta_1 \shortmid \ldots \shortmid \delta_k) \cdot \kappa(\delta_{k+1}) = \sigma_1 \cdot \ldots \cdot \sigma_r \cdot \sigma_{r+1} \cdot \ldots \cdot \sigma_{r+s}$, where $\kappa(\delta_{k+1}) = \sigma_{r+1} \cdot \ldots \cdot \sigma_{r+s}$. Since $r \geq k$ and $s \geq 1$, we have that $r + s \geq k + 1$. □

8.2 Examples

Example 8.1. *Consider the simple configuration represented in Fig. 1, where both τ and κ are deterministic maps. Let $p^\star(\boldsymbol{\sigma}_1) = 0.2$ and $p^\star(\boldsymbol{\sigma}_2) = p^\star(\boldsymbol{\sigma}_3) = 0.4$, with $p^\star(\boldsymbol{\sigma}_i) = 0$ for $i > 3$. For $q^\star = \tau p^\star$, we have, therefore, $q^\star(\boldsymbol{\delta}_1) = 0.2$, $q^\star(\boldsymbol{\delta}_2) = 0$, and $q^\star(\boldsymbol{\delta}_3) = 0.8$, with $q^\star(\boldsymbol{\delta}_i) = 0$ for $i > 3$, and hence $\kappa\tau p^\star(\boldsymbol{\sigma}_1) = 0 \neq 0.2$, $\kappa\tau p^\star(\boldsymbol{\sigma}_2) = 0.2 \neq 0.4$, and $\kappa\tau p^\star(\boldsymbol{\sigma}_1) = 0.8 \neq 0.4$. Assuming $\{q_n\}$ is a consistent estimator of $q^\star$, the pushforward of q_n through κ (i.e., κq_n) would result in an inconsistent estimation of $p^\star$. Notice that the consistency of the tokenizer is relative to the distribution. Relative to a different distribution p in Σ^*, where, for instance, $p(\boldsymbol{\sigma}_1) = p(\boldsymbol{\sigma}_2) = 0$ and $p = p^\star$ otherwise, the tokenizer specified in Fig. 1 is consistent.*

Example 8.2. *Take, for instance, a strictly bijective tokenizer such as BPE or WordPiece, with κ performing concatenation of the token maps in the usual way. Let $\Sigma = \{\mathtt{t}, \mathtt{h}, \mathtt{e}\}$ and $\Delta = \{\mathtt{t}, \mathtt{h}, \mathtt{e}, \mathtt{th}, \mathtt{he}\}$. In this minimal configuration, it is easy to see that $\kappa(\mathtt{t} \shortmid \mathtt{h} \shortmid \mathtt{e}) = \kappa(\mathtt{t} \shortmid \mathtt{he}) = \kappa(\mathtt{th} \shortmid \mathtt{e}) = \mathtt{t} \cdot \mathtt{h} \cdot \mathtt{e} \in \Sigma^*$. However, BPE or WordPiece being strictly bijective tokenizers, τ can only map the value of κ to at most one of the latter's arguments, say $\tau(\mathtt{t} \cdot \mathtt{h} \cdot \mathtt{e}) = \mathtt{th} \shortmid \mathtt{e}$. We then have that $\tau(\kappa(\mathtt{t} \shortmid \mathtt{he})) \neq \mathtt{t} \shortmid \mathtt{he}$ (and likewise for $\mathtt{t} \shortmid \mathtt{h} \shortmid \mathtt{e}$). If the estimator happens to place nonzero probability mass on any of the latter two token sequences, the model will exhibit spurious ambiguity.*

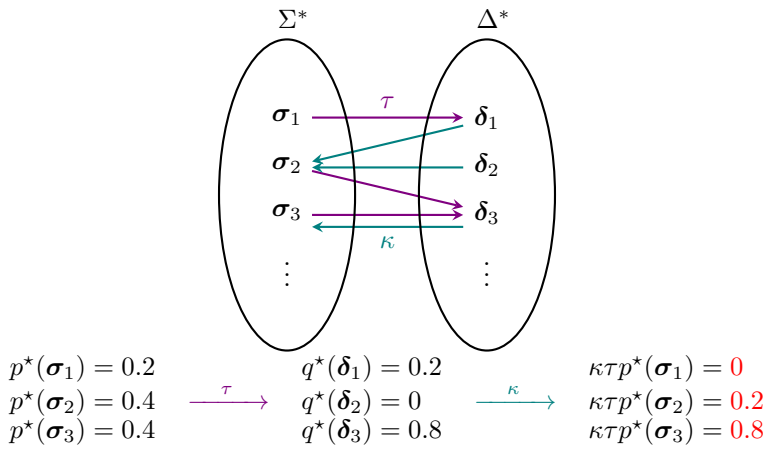


Figure 1: Example of an inconsistent tokenizer with τ represented in violet and κ represented in teal.

Example 8.3. *Let $\Sigma = \Delta = \{\texttt{0}, \texttt{1}\}$, and define $\mathcal{T}_{unc} = (\tau_{unc}, \kappa_{unc})$ as a deterministic model in the following way:*

$$\tau_{unc}(\boldsymbol{\sigma}) = \begin{cases} \boldsymbol{\sigma}\shortmid\texttt{1}, & \textit{if } \boldsymbol{\sigma} \textit{ describes a valid} \\ & \textit{Turing Machine followed by} \\ & \textit{an input for which it halts.} \\ \boldsymbol{\sigma}\shortmid\texttt{0}, & \textit{otherwise.} \end{cases} \qquad \kappa_{unc}(\boldsymbol{\delta}) = \begin{cases} \varepsilon, & \textit{if } \boldsymbol{\delta} \in \Delta. \\ \boldsymbol{\sigma}, & \textit{otherwise, where } \boldsymbol{\delta} = \boldsymbol{\sigma}\shortmid\delta. \end{cases}$$

Significantly, $\mathcal{T}_{unc}$ is not only well-defined but also exact and therefore consistent for any language model p over Σ^*. However, τ_{unc} is famously an uncomputable function, and hence $\mathcal{T}_{unc}$ is an uncomputable tokenizer.