arXiv:2506.06446v2 [cs.CL] 30 Jan 2026

Tokenization Multiplicity Leads to Arbitrary Price Variation in LLM-as-a-service

Ivi Chatzi[1], Nina Corvelo Benz[2], Stratis Tsirtsis[3], and Manuel Gomez-Rodriguez[1]

[1]Max Planck Institute for Software Systems, Kaiserslautern, Germany
{ichatzi, manuel}@mpi-sws.org

[2]Max Planck Institute of Biochemistry, Martinsried, Germany
corvelo@biochem.mpg.de

[3]Hasso Plattner Institute, Potsdam, Germany
stratis.tsirtsis@hpi.de

Abstract

Providers of LLM-as-a-service have predominantly adopted a simple pricing model: users pay a fixed price per token. Consequently, one may think that the price two different users would pay for the same output string under the same input prompt is the same. In our work, we show that, surprisingly, this is not (always) true. We find empirical evidence that, particularly for non-english outputs, both proprietary and open-weights LLMs often generate the same (output) string with multiple different tokenizations, even under the same input prompt, and this in turn leads to arbitrary price variation. To address the problem of tokenization multiplicity, we introduce canonical generation, a type of constrained generation that restricts LLMs to only generate canonical tokenizations—the unique tokenization in which each string is tokenized during the training process of an LLM. Further, we introduce an efficient sampling algorithm for canonical generation based on the Gumbel-Max trick. Experiments on a variety of natural language tasks demonstrate that canonical generation is comparable to standard generation in terms of performance and runtime, and it solves the problem of tokenization multiplicity.

1 Introduction

Imagine you run an online service that offers AI-powered translation to help tourists seamlessly navigate websites in languages other than their mother tongue. Behind the scenes, for each website the users visit, your service sends the website's text to a provider of an LLM and asks their model to generate the translation in the mother tongue of the user. One day, reviewing your costs, you notice something strange: Hans and Emma, two users of your service, visited the same page on a website with geographical fun facts, requesting a German translation of the sentence

"The Acari River is a river of Minas Gerais state in southeastern Brazil".

Both of them received the same translated text down to the last character, yet, you were charged different amounts for each by the LLM provider. How can identical results have different costs?

The computational resources required to operate state-of-the-art large language models are too large for most (enterprise) users to run them locally (Narayanan et al., 2021; Jiang et al., 2024). As a consequence, users are relying on a growing market of cloud-based providers who offer access to such models via an application programming interface (API) (Sun et al., 2022; La Malfa et al., 2024; Pais et al., 2022; Liagkou et al., 2024).

The way that providers of LLM-as-a-service have come to charge the use of their models is heavily dependent on one of the most distinctive technical characteristics shared by most, if not all, state-of-the-art models—they process and generate information in discrete units called tokens.[1] In particular, since the computational resources required to process an input prompt and generate a response are directly proportional to the number of tokens involved (Samsi et al., 2023; Fernandez et al., 2025), providers have predominantly adopted a straightforward pay-per-token pricing model—users pay a fixed price per token.

Under the pay-per-token pricing model, one may naturally assume that, if i) two users submit the same prompt to a provider, ii) the provider feeds the prompt into the same model, and iii) both users receive exactly the same output string, then, they will pay the same price. In our work, we show that, surprisingly, this assumption does not hold. We find empirical evidence that, particularly for non-english outputs, both proprietary and open-weights LLMs often generate the same (output) string with different tokenizations, even under the same input prompt, and this multiplicity of tokenizations in turn leads to price variation. Moreover, since users derive value from the text that an output token sequence represents, rather than the token sequence itself, we argue that this price variation is arbitrary and undesirable.

To address the problem of tokenization multiplicity, we introduce canonical generation, a type of constrained generation that restricts LLMs to only generate canonical tokenizations—the unique tokenization in which each string is tokenized during the training process of an LLM (Geh et al., 2024). In doing so, we also introduce an efficient sampling algorithm for canonical generation based on the Gumbel-Max trick that leverages the following theoretical result, which may be of independent interest: to generate a canonical tokenization, a model needs to generate (partial) canonical tokenizations at each step of the generation process underpinning its functioning.

In addition to solving the problem of tokenization multiplicity, we show that, in comparison with standard generation, the distribution of token sequences generated by canonical generation is provably closer to the true distribution of token sequences used during training and, in practice, the performance and runtime of canonical generation are comparable to standard generation. We have released all code and data used in our experiments at: https://github.com/Networks-Learning/Tokenization-Multiplicity.

Further related work. Our work builds upon further related work on tokenization in LLMs, multilingual LLMs, and the economics of LLM-as-a-service.

The study of tokenization has a rich history in natural language processing (Palmer, 2000; Jurafsky and Martin, 2025). More recently, in the context of LLMs, there has been a renewed interest in formalizing tokenization and analyzing its properties (Gastaldi et al., 2025; Phan et al., 2024; Rajaraman et al., 2025), with the Byte-Pair Encoding (BPE) tokenization algorithm in particular receiving increased attention (Berglund and van der Merwe, 2023; Zouhar et al., 2023; Kozma and Voderholzer, 2024). The impact of tokenization on LLMs has also been studied empirically (Hou et al., 2023; Athiwaratkun et al., 2024) in generation tasks involving foreign languages (Fujii et al., 2023), translation (Domingo et al., 2019), arithmetic (Singh and Strouse, 2024), mental health (Liu et al., 2023), and privacy (Kharitonov et al., 2021; Petrov et al., 2023), among others. Moreover, there is a recent line of work studying the impact of non-canonical tokenizations on text perplexity calculations (Cao and Rimell, 2021; Chirkova et al., 2023; Geh et al., 2024; Vieira et al., 2025a; Giulianelli et al., 2024), safety guidelines (Geh et al., 2025), downstream tasks (Zheng et al., 2025), and image watermarking (Jovanović et al., 2025), as well as an effort to circumvent non-canonical tokenizations occuring as a consequence of unusual input prompt endings (Guidance, 2023). Within this line of work, the work most closely related to ours is by Vieira et al. (2025b), who have also independently proposed canonical generation[2]. However, their work does not provide empirical evidence of tokenization multiplicity and its influence on the pay-per-token pricing model as we do, their sampling algorithms for canonical generation are computationally less efficient compared to ours, and their experimental evaluation does not demonstrate that canonical generation is comparable to standard generation in terms of performance across natural language tasks.

There has been extensive research on the capabilities of multilingual LLMs, including their performance on downstream tasks (Fujii et al., 2023; Zhang et al., 2022; Rust et al., 2021) as well as their safety

---

[1] Tokens are (sub-)words, symbols and numbers that make up sentences and paragraphs.

[2] A preliminary version of our work and their work were posted on arXiv within three days of each other.

vulnerabilities (Shen et al., 2024; Wang et al., 2024; Deng et al., 2024; Dong et al., 2025) across different languages. Within this research, the work most related to ours is by Ahia et al. (2023), who have shown that multilingual LLMs require more tokens to generate text of similar meaning in minority languages than in english. Our work complements their work by providing empirical evidence that multilingual LLMs suffer from tokenization multiplicity in minority languages.

The rapidly growing literature on the economics of LLM-as-a-service (Mahmood, 2024; Laufer et al., 2024; Cai et al., 2025; Saig et al., 2024; Bergemann et al., 2025; Sun et al., 2025a,b; Artola Velasco et al., 2025; Velasco et al., 2026) has predominantly focused on understanding the incentives providers may have to act strategically at the expense of users. Within this literature, the work most closely related to ours is by Artola Velasco et al. (2025), who show that token multiplicity enables an unfaithful provider to strategize and misreport the tokenization of an output generated by a model they serve without raising suspicion. Moreover, to eliminate the incentive to strategize, they propose an incentive-compatible pay-per-character pricing model, which in turn also eliminates the price variation due to tokenization multiplicity. However, in their work, they do not empirically demonstrate that token multiplicity can occur in practice, and it can lead to arbitrary price variation even under faithful providers, as we do in our work. Further, it is worth noting that the pay-per-character pricing model makes a provider's profit margin vary across tokens, and this may discourage its practical adoption.

2 Preliminaries

In this section, we first define and formally characterize (deterministic) tokenizers and canonical tokenizations. Then, we briefly review the aspects of LLM training and generation that are relevant for our work.

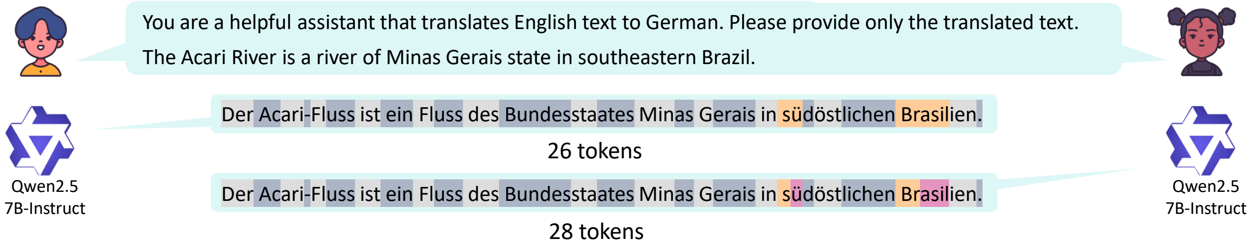
Tokenizers and canonical tokenizations. Tokenizers are tools that operate on sequences of characters (*i.e.*, strings) and sequences of tokens, and can transform one type into the other. Formally, let Σ be a finite set of characters and Σ^+ be the set of all finite strings that can be created using the characters in Σ. Similarly, let V be a finite set of tokens, which we will refer to as the vocabulary, and V^+ be the set of all finite token sequences that can be created using the tokens in V. Then, a tokenizer $\mathcal{T}$ is characterized by a tuple $\mathcal{T} := (\Sigma, V, \texttt{enc}, \texttt{dec})$, where $\texttt{enc} : \Sigma^+ \rightarrow V^+$ is an encoder, which transforms strings to token sequences, and $\texttt{dec} : V^+ \rightarrow \Sigma^+$ is a decoder, which transforms token sequences to strings.

Let $\boldsymbol{\sigma}$ be a string and $\mathbf{s} \in V^+$ be a token sequence such that $\texttt{dec}(\mathbf{s}) = \boldsymbol{\sigma}$, then, we will say that $\mathbf{s}$ is a (valid) tokenization of the string $\boldsymbol{\sigma}$. Here, note that there may be multiple tokenizations of a string $\boldsymbol{\sigma}$, that is, there may exist $\mathbf{s}, \mathbf{s}' \in V^+$ such that $\mathbf{s} \neq \mathbf{s}'$ and $\texttt{dec}(\mathbf{s}) = \texttt{dec}(\mathbf{s}') = \boldsymbol{\sigma}$. However, given a string $\boldsymbol{\sigma}$, the encoder deterministically picks a single tokenization $\texttt{enc}(\boldsymbol{\sigma})$ among all tokenizations of $\boldsymbol{\sigma}$, which is often called the canonical tokenization (Geh et al., 2024). For details on the most commonly used tokenization algorithms—BPE (Gage, 1994; Sennrich et al., 2016), Unigram (Kudo, 2018) and Wordpiece (Song et al., 2021)—refer to Appendix A.

LLM training and generation. During training, an LLM learns to predict the next token in canonical sequences of tokens derived from raw text using a tokenizer. More formally, let $p_{\mathbf{s}} = P[T \mid \boldsymbol{S} = \mathbf{s}]$ denote the true distribution of the random variable $T \in V$, representing the next token given a (partial) token sequence $\mathbf{s} \in V^+$. Then, the goal of LLM training is (typically) to minimize the (cross-entropy) loss between the model's predicted distribution $d_{\mathbf{s}} \in \Delta(V)$ and the true next-token distribution $p_{\mathbf{s}}$.

During generation, an LLM takes as input a prompt sequence $\mathbf{s}_q \in V^+$ and responds with an output sequence $\mathbf{s} \in V^+$, generated using an autoregressive process. At each time step of the process, the LLM first takes as input the concatenation of the prompt sequence $\mathbf{s}_q$ and the (partial) output sequence $\mathbf{s}$, and generates a distribution over tokens $d_{\mathbf{s}_q \mid \mathbf{s}} \in \Delta(V)$. Then, it samples the next token t from the distribution $d_{\mathbf{s}_q \mid \mathbf{s}}$ and appends the token t to the output sequence $\mathbf{s}$. The process continues until a special end-of-sequence token is sampled. For the remainder of this paper, we will omit the prompt sequence $\mathbf{s}_q$ from the notation and write $d_{\mathbf{s}}$ and $p_{\mathbf{s}}$ for brevity.

Importantly, since LLMs are trained on finite data, the support of the distribution $d_{\mathbf{s}}$ may differ from the respective true distribution $p_{\mathbf{s}}$. As a result, it is possible for an LLM to generate a non-canonical token



You are a helpful assistant that translates English text to German. Please provide only the translated text.
The Acari River is a river of Minas Gerais state in southeastern Brazil.

Der Acari-Fluss ist ein Fluss des Bundesstaates Minas Gerais in südöstlichen Brasilien.

Der Acari-Fluss ist ein Fluss des Bundesstaates Minas Gerais in südöstlichen Brasilien.

Figure 1: **Translation task example.** The top box shows the input prompt, which consists of a translation instruction and the accompanying Wikipedia text to be translated. The latter two boxes show two outputs generated by `Qwen2.5-7B-Instruct` as response to the input prompt, corresponding to the same string but with two different tokenizations.

sequence, even if it has encountered no such sequences during training (Cao and Rimell, 2021; Chirkova et al., 2023; Vieira et al., 2025a; Giulianelli et al., 2024; Geh et al., 2025).

3 Can Hans and Emma Receive Different Tokenizations for the Same Output String?

We answer this question affirmatively. Using three simple natural language tasks, we demonstrate that tokenization multiplicity can occur in both proprietary and open-weights LLMs. Specifically, we focus on translation (as in the example of Hans and Emma in Section 1), spell checking, and rephrasing. The motivation for these choices is that, in all these tasks, an LLM has to process and rewrite a given input text and is therefore more likely to generate the same output string despite the randomness in its generation.

For each of the three tasks, we construct 100 input prompts using short texts from Wikipedia.[3] For the translation task, each prompt consists of a (system) instruction to translate a short text written in English to one of 5 target languages, followed by the respective Wikipedia text in the English language. For the other two tasks, each prompt consists of a (system) instruction to spell check or rephrase short texts written in 6 different languages (including English) and, for the spell checking task, we first introduce a small number of typos in each Wikipedia text by replacing a few latin characters with other latin characters selected at random. To simulate scenarios where different users ask an LLM to perform the same task on the same input text, we feed each input prompt to the LLM 100 times, keeping all parameters identical except for the random seed used during generation. We then focus on (the existence of) pairs of outputs whose strings are identical but their tokenization lengths differ.

As a starting point, in Figure 1, we illustrate the translation task that we consider in our experiments using as an example a pair of outputs generated by `Qwen2.5-7B-Instruct` (Yang et al., 2024) to the input text introduced at the beginning of Section 1. We observe that, while both output pairs correspond to the same string, Hans's output consists of 26 tokens while Emma's output consists of 28 tokens due to a difference in the generated tokenization of the words “südöstlichen” and “Brasilien”. Importantly, these tokenization differences cause the latter output to be 7.7% more expensive than the former under pay-per-token pricing. For similar examples illustrating the spell checking and rephrasing tasks, refer to Appendix E.

Further, we proceed to quantify the frequency and magnitude of such inconsistencies in pricing across different tasks, LLMs, and languages. In our experiments, we consider both proprietary LLMs from the `gpt`, `gemini`, and `claude` families and open-weights LLMs from the `Llama` and `Qwen` families. For brevity, in the remainder of the section, we use shortened names to refer to each model, and we provide the full names of all models in Table 3 in Appendix F, along with additional details about the hardware, data, and APIs we use.[4]

---

[3] https://dumps.wikimedia.org/

[4] For (proprietary) `gpt`, `gemini`, and `claude` models, we used the official LLM-as-a-service APIs from OpenAI, Google, and Anthropic, respectively. For open-weights models, we ran all experiments locally, however, note that a user without access to specialized hardware would normally also access them via LLM-as-a-service APIs from third-party providers.

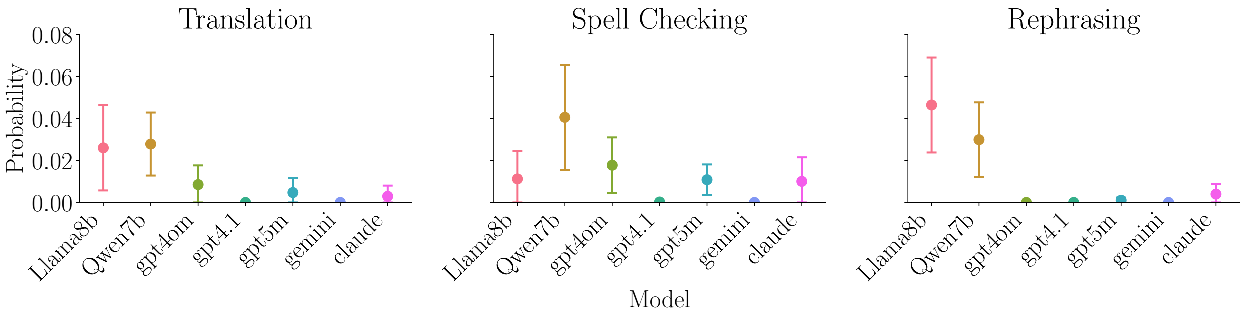


Figure 2: **Probability of tokenization multiplicity.** The plots show the empirical probability that the length of two output tokenizations to the same input prompt differ, conditioned on the output strings being the same. Each panel corresponds to one of the three tasks we consider in our experiments involving outputs in the German language. Across all panels, error bars represent 95% confidence intervals resulting from 100 input prompts.

We first look into the (conditional) probability that, once two users receive the same output string from the model, the lengths of the two tokenizations differ. Formally, let $\mathbf{S}$ and $\mathbf{S}'$ denote the random variables corresponding to tokenizations received by two different users as a response after providing the same prompt, *i.e.*, $Q = Q'$. Our goal is to estimate the quantity $P(\texttt{len}(\mathbf{S}) \neq \texttt{len}(\mathbf{S}') \mid \texttt{dec}(\mathbf{S}) = \texttt{dec}(\mathbf{S}'), Q = Q')$. To this end, for each prompt, we count the number of output pairs whose strings match but tokenization lengths differ as a fraction of all output pairs whose strings match, and we take the average across prompts. Note that, to perform fair comparisons across models, we intentionally focus on cases where there are differences in the tokenization length rather than the tokenization itself, since the generated tokenization is not observable under all APIs.[5] Figure 2 summarizes the results for German language, which show that tokenization multiplicity can indeed occur both when using open-weights and proprietary models. In particular, for all models except `gemini`, we find cases of tokenization multiplicity in at least one task and, for open-weights models (*i.e.*, `Llama8b` and `Qwen7b`), we find that tokenization multiplicity occurs regularly across all three tasks. Refer to Appendix G.1 for qualitatively similar results in other languages. In this context, note that the `gemini` model did generate a few (<1%) non-canonical outputs, which suggests that this model may also present tokenization multiplicity, but we did not observe it due to the finite sample size of our experiments.[6]

Next, we focus on outputs where tokenization multiplicity does occur and look into the magnitude of the variation in tokenization lengths, which directly determines the variation in prices charged to users under pay-per-token pricing. Specifically, for each output string $\boldsymbol{\sigma}$ where there are at least two tokenizations with different length, we measure the relative difference in length (and hence price) between the shortest tokenization $\mathbf{s}_{min}$ and the longest tokenization $\mathbf{s}_{max}$, *i.e.*, $\left(\texttt{len}\left(\mathbf{s}_{max}\right) - \texttt{len}\left(\mathbf{s}_{min}\right)\right) / \texttt{len}\left(\mathbf{s}_{min}\right)$. Figure 3 summarizes the results for German language, which show that, whenever tokenization multiplicity occurs, a user may pay up to 15% higher price in comparison with another user for the same output string, with similar levels of variation across both open-weights and proprietary models. Refer to Appendix G.1 for qualitatively similar results in other languages.

Further, we analyze how the prevalence of tokenization multiplicity changes depending on the language of the output. To this end, out of the 100 input prompts we construct per language and task, we measure the number of prompts for which a model (here, `gpt5m`) generates at least two outputs with the same string but different tokenization lengths. Figure 4 summarizes the results, which show that, across all three tasks, tokenization multiplicity is more prevalent in minority languages. For example, even though ChatGPT

---

[5] The API services for `gpt5m`, `gemini` and `claude` return only the output string without disclosing the tokenization that was generated. In those cases, it is still possible to identify differences in tokenization length, which is always disclosed since pricing directly depends on it.

[6] Refer to Appendix G.1 for additional results showing the percentage of non-canonical outputs across all models and tasks.

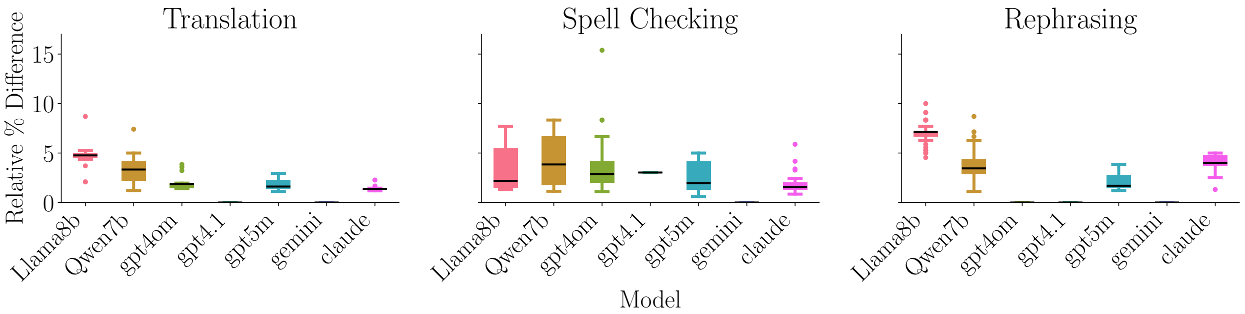


Figure 3: **Magnitude of price variation.** The plots show the empirical distribution of the relative difference in length between the longest and shortest tokenization of each output string, across all outputs where tokenization multiplicity occurs. Each panel corresponds to one of the three tasks we consider in our experiments involving outputs in the German language. Across all panels, box plots show the quartiles of the respective distributions with black horizontal lines representing median values.

officially supports languages such as Turkish or Swahili,[7] we observed that at least 7% of the input prompts for these two languages led `gpt5m` to generate identical outputs with different prices. For qualitatively similar results with other models, refer to Appendix G.1.

To study the prevalence and magnitude of tokenization multiplicity, we have focused so far on pairs of short outputs corresponding to exactly the same string, since this allows us to attribute any price variation exclusively to token multiplicity. In this context, one may think that, on pairs of longer outputs with multiple partial string matches, tokenization multiplicity may not lead to significant price variation if the difference in tokenization length across partial matches cancels out. However, in what follows, we provide empirical evidence that the difference in tokenization length is not independent across partial matches and, as a consequence, the price variation due to token multiplicity may be more pronounced in longer outputs. In particular, in outputs generated by `gpt4om` to the translation task from English to German, but using longer Wikipedia texts (refer to Appendix F for details regarding the experimental setup), we find that, if a word is generated with a non-canonical tokenization, then, subsequent occurrences of the word are typically generated using that same tokenization. More specifically, we observe that, in 88% of those cases, all subsequent occurrences (up to 10) are generated using the same non-canonical tokenization. In contrast, in 10% of the remaining cases, all subsequent occurrences of the word are generated using the canonical tokenization and, in the last 2% of cases, subsequent occurrences use a mix of both canonical and non-canonical tokenizations. As an immediate consequence, it is easy to find examples of long, similar outputs containing multiple repetitions of the same word with different but consistent tokenizations, as shown in Figure 6 in Appendix E.

4 Avoiding Tokenization Multiplicity through Canonical Generation

To solve the problem of tokenization multiplicity, we introduce canonical generation, a type of constrained generation that restricts LLMs to generate each output string with only its canonical tokenization. In this context, we argue that, if one were to pick a single tokenization for each output string, the canonical tokenization presents itself as the most natural choice. This is because, during the training of an LLM, all strings in the training data are first encoded canonically using a tokenizer, and the LLM is then trained to complete these canonical token sequences.

In what follows, we first show that, for an output token sequence generated by an LLM to be canonical, the partial token sequences generated at each step of generation must also be canonical. Building upon this result, we then introduce an efficient and easy-to-implement sampling algorithm for canonical generation based on the Gumbel-Max trick (Huijben et al., 2022). Finally, we conclude by analyzing both theoretically

---

[7] https://help.openai.com/en/articles/8357869

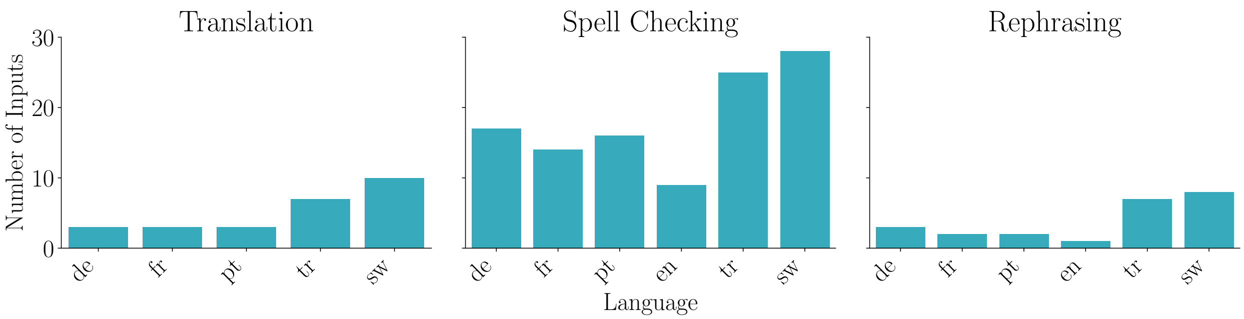


Figure 4: **Tokenization multiplicity across languages.** The plots show the number of inputs prompts for which we observe at least two outputs given by `gpt5m` with the same string but different tokenization lengths. Each panel corresponds to one of the three tasks we consider in our experiments and pairs of letters on the x-axis correspond to different languages. Refer to Table 2 in Appendix F for details regarding the languages we use and to Appendix G.1 for qualitatively similar results using other models.

and empirically the quality of the outputs generated by canonical generation against those generated by standard generation, as well as the runtime of our sampling algorithm.

4.1 Subsequences of Canonical Token Sequences Must Also Be Canonical

In this section, we establish our main theoretical result, which shows that the most commonly used tokenizers—BPE, Unigram, and Wordpiece—are *non-recovering* tokenizers.

Definition 1. *A tokenizer $\mathcal{T} = (\Sigma, V, \texttt{enc}, \texttt{dec})$ is called non-recovering if it holds that, for any non-canonical token sequence $\mathbf{s} \in V^+$ according to $\mathcal{T}$ and any token $t \in V$, $\mathbf{s} \shortmid t$ is also non-canonical.*

More formally, we have the following theorem:[8]

Theorem 2. *BPE-, Unigram- and Wordpiece-based tokenizers are non-recovering.*

This theorem immediately implies that an output token sequence is canonical if and only if the partial token sequences generated at each step of the generation process are canonical. Moreover, this theorem also provides a plausible explanation for the empirical observation that the likelihood that an LLM generates non-canonical output sequences increases with the length of the sequence (Geh et al., 2024). This is because, since sampling a "non-canonical token" once during the generation process is sufficient to render the output token sequence non-canonical, it is natural that the chances of this to happen increase with the number of sampled tokens.

In this context, we find it rather surprising that the above theorem holds for BPE, Unigram and Wordpiece since these (deterministic) tokenizers use fundamentally different tokenization techniques: BPE uses a rule based approach, Unigram maximizes the probability of the tokenization, and Wordpiece greedily encodes the text to minimize the number of tokens.

4.2 An Efficient Sampling Algorithm for Canonical Generation

Building upon Theorem 2, we now introduce canonical generation, along with an efficient sampling algorithm to implement it. The core principle of canonical generation is to ensure that the sampled tokens at all steps of the generation are such that the respective (partial) output remains canonical. To this end, at each step, it sets the probability of a subset of tokens to zero—those that, when appended to a partial output sequence,

---

[8] The proof for Theorem 2 can be found in Appendix B. In Appendix C, we also prove that, under mild conditions, the theorem holds when using pretokenization.

Algorithm 1 Canonical Generation via Gumbel-Max Sampling

```
Require: next-token distribution $d_{\mathbf{s}}$
  $u_t \sim Gumbel(0, 1)$ for all $t \in V$
  for $t \in V$ in decreasing order of $\log(d_{\mathbf{s}}(t)) + u_t$ do
      if $s \shortmid t$ is canonical then
          return t
      end if
  end for
```

would result in a non-canonical token sequence—and redistributes their probability mass to the remaining tokens proportionally to their original probability mass.

Formally, let $d_{\mathbf{s}}$ denote the next-token distribution generated by the LLM given a partial output token sequence $\mathbf{s}$ and let $d_{\mathbf{s}}(t)$ denote the probability of sampling a token t from this distribution. Given the partial output token sequence $\mathbf{s}$, an LLM using canonical generation draws the next token in the generation process from a *canonicalized* next-token distribution

$$\tilde{d}_{\mathbf{s}}(t) := \begin{cases} d_{\mathbf{s}}(t)/Z & \text{if } \mathbf{s} \shortmid t \text{ is canonical} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $Z = \sum_{t \in V:\ \mathbf{s} \shortmid t \text{ is canonical}} d_{\mathbf{s}}(t)$ is a normalization constant that ensures that $\tilde{d}_{\mathbf{s}}$ is a valid probability distribution. In that context, note that redistributing the probability mass of tokens that would lead to non-canonical token sequences proportionally to the original probabilities $d_{\mathbf{s}}(t)$ is a natural choice we make, inspired by other popular strategies for (stochastic) generation, such as top-k and top-p sampling (Holtzman et al., 2020), and constrained generation (Beurer-Kellner et al., 2024).

Next, we introduce an efficient and easy-to-implement algorithm to sample from the canonicalized next-token distribution $\tilde{d}_{\mathbf{s}}$, which avoids explicitly computing the entire distribution $\tilde{d}_{\mathbf{s}}$. The algorithm starts by sampling a value $u_t \sim \text{Gumbel}(0, 1)$ from a Gumbel distribution for each token $t \in V$. Then, it ranks the tokens in decreasing order with respect to the perturbed log-probability $\log(d_{\mathbf{s}}(t)) + u_t$. Finally, it returns the token t with the largest value of $\log(d_{\mathbf{s}}(t)) + u_t$ such that $\mathbf{s} \shortmid t$ is canonical. The overall procedure, summarized in Algorithm 1, relies on a property of the Gumbel-Max trick (Maddison et al., 2014; Huijben et al., 2022), which states that the argmax operation over a constrained subset of categorical outcomes is equivalent to sampling from a categorical distribution with zero probability for all outcomes outside the subset, and with the probabilities of the outcomes in the subset scaled proportionally to their original probabilities, as shown in Eq. 2 in Maddison et al. (2014). Hence, it readily holds that Algorithm 1 returns a valid sample from the canonicalized next-token distribution $\tilde{d}_{\mathbf{s}}$ defined in Eq. 1, *i.e.*,

$$\operatorname*{argmax}_{t \in V:\ \mathbf{s} \shortmid t \text{ is canonical}} \{\log(d_{\mathbf{s}}(t)) + u_t\} \sim \tilde{d}_{\mathbf{s}}.$$

Further, it is worth highlighting that, in contrast to computing the canonicalized next-token distribution $\tilde{d}_{\mathbf{s}}$, which requires evaluating the canonicity of $|V|$ token sequences, Algorithm 1 requires only a few evaluations of canonicity. This is because, in practice, LLMs tend to generate mostly canonical token sequences (Geh et al., 2024), hence, the probabilities $d_{\mathbf{s}}(t)$ generated by an LLM for tokens t that lead to non-canonical sequences $\mathbf{s} \shortmid t$ are usually small. More specifically, let $d_{\mathbf{s}}(\text{canonical})$ be the probability mass on the subset of tokens that lead to canonical sequences, *i.e.*, $d_{\mathbf{s}}(\text{canonical}) = \sum_{t \in V:\ \mathbf{s} \shortmid t \text{ is canonical}} d_{\mathbf{s}}(t)$, then Algorithm 1 requires, in expectation, fewer than $1/d_{\mathbf{s}}(\text{canonical})$ evaluations of canonicity before successfully sampling the next token. That is because, unlike (independent) rejection sampling from $d_{\mathbf{s}}$, which would require in expectation exactly $1/d_{\mathbf{s}}(\text{canonical})$ evaluations of canonicity until a token that leads to a canonical sequence is successfully sampled, our approach never checks the same token twice, which results in an increase in the success probability of sampling a token that leads to a canonical sequence after each failed attempt.[9]

---

[9] The number of evaluations of canonicity in rejection sampling is distributed according to a geometric distribution with success probability $d_{\mathbf{s}}(\text{canonical})$ resulting in $1/d_{\mathbf{s}}(\text{canonical})$ evaluations in expectation until a successful sample.

Task	Metric	Llama8B		Qwen7B	
		Standard	Canonical	Standard	Canonical
Translation	Quality score	0.72 ± 0.02	0.70 ± 0.02	0.73 ± 0.01	0.71 ± 0.01
	Time per token (s)	0.019	0.020	0.018	0.019
	Non-canonicity rate	6%	-	18%	-
Spell Checking	$1 -$ edit distance	0.62 ± 0.04	0.61 ± 0.04	0.74 ± 0.04	0.72 ± 0.04
	Time per token (s)	0.020	0.023	0.018	0.018
	Non-canonicity rate	10%	-	19%	-
Rephrasing	Cosine similarity	0.84 ± 0.02	0.84 ± 0.02	0.89 ± 0.02	0.88 ± 0.02
	Time per token (s)	0.020	0.020	0.018	0.020
	Non-canonicity rate	6%	-	5%	-
MGSM	Accuracy	0.37 ± 0.06	0.37 ± 0.06	0.63 ± 0.05	0.62 ± 0.06
	Time per token (s)	0.020	0.021	0.018	0.020
	Non-canonicity rate	22%	-	29%	-

Table 1: **Performance, (generation) time per token, and non-canonicity rate.** The results comprise pairs of outputs generated with standard and canonical generation in German language under the same source of randomness. For the time per token, confidence intervals are not shown, as they were all smaller than 10^{-4}.

The simplest way to test whether the sequence $\mathbf{s} \shortmid t$ is canonical is to compute and check if $\texttt{enc}(\texttt{dec}(\mathbf{s} \shortmid t)) = \mathbf{s} \shortmid t$. For BPE-based tokenizers however, it has been shown that it is sufficient to test if $\texttt{enc}(\texttt{dec}(t_{\text{last}} \shortmid t)) = t_{\text{last}} \shortmid t$, where t_{last} is the final token in $\mathbf{s}$. In fact, a recently proposed efficient algorithm to test whether $\mathbf{s} \shortmid t$ is canonical only partially applies the BPE algorithm to $\texttt{dec}(t_{\text{last}} \shortmid t)$ (Vieira et al., 2025b; Hayase et al., 2025).

4.3 Performance of Canonical Generation

We first show that, in comparison with standard generation, the distribution of tokens generated by canonical generation is provably closer to the true distribution of sequences that the LLM has seen during training. Formally, let p denote the true distribution over token sequences $\mathbf{s} \in V^+$ used during training, for which note that $p(\mathbf{s}) = 0$ holds for all sequences $\mathbf{s}$ that are non-canonical. Moreover, let d denote the distribution over token sequences that the LLM generates using standard generation, and $\tilde{d}$ the distribution over token sequences that the LLM generates using canonical generation, that is, sampling from the canonicalized next-token distribution $\tilde{d}_{\mathbf{s}}$ given by Eq. 1 at each step of the generation process. Then, the following theorem shows that p is likely to be closer to $\tilde{d}$ than d in terms of KL-divergence, a result independently established in Proposition 3 of Vieira et al. (2025b):[10]

Theorem 3. *Let d be absolutely continuous*[11] *with respect to p. Moreover, assume that there exist $\mathbf{s} \in V^+$ and $t_1, t_2 \in V$ such that $\mathbf{s} \shortmid t_1$ is non-canonical with $d(\mathbf{s} \shortmid t_1) > 0$ and $\mathbf{s} \shortmid t_2$ is canonical with $p(\mathbf{s} \shortmid t_2) > 0$ and $d(\mathbf{s} \shortmid t_2) > 0$. Then, it holds that*

$$KL(p, \tilde{d}) < KL(p, d). \tag{2}$$

In simpler terms, the two conditions under which canonical generation brings the output token sequences closer to the true distribution are that (i) there exist non-canonical token sequences with positive probability of being generated under d so that their probability mass can be redistributed, and (ii) there exist canonical

---

[10] The proof for Theorem 3 can be found in Appendix D.

[11] Absolute continuity is required for the KL-divergence to be well defined, *i.e.*, we require that $d(\mathbf{s}) = 0$ implies that $p(\mathbf{s}) = 0$ for all $\mathbf{s} \in V^+$.

token sequences with positive probability under d and p so that the redistribution of probability mass in $\tilde{d}$ is beneficial. To understand the intuition behind Theorem 3, note that, by using canonical generation (*i.e.*, sampling from $\tilde{d}$ instead of d), the probability that an LLM generates non-canonical token sequences becomes zero, and the probability that it generates any other (canonical) token sequence increases under $\tilde{d}$. Further, since only canonical token sequences have positive probability under the true distribution p, this redistribution of probability mass from non-canonical token sequences to canonical ones can only bring the distribution $\tilde{d}$ closer to the true distribution p compared to d.

On the flip side, it is important to clarify that a similar property does not necessarily hold for the respective distributions over strings. That is, using canonical generation, the distribution of output strings, resulting from decoding the output token sequences, is not guaranteed to be closer (in terms of KL-divergence) to the true distribution of output strings used during training. Formally, let $p_{\texttt{dec}} = P_{\mathsf{s}\sim p(\mathsf{s})}[\texttt{dec}(\mathsf{s})]$ be the true distribution over strings, $d_{\texttt{dec}} = P_{\mathsf{s}\sim d(\mathsf{s})}[\texttt{dec}(\mathsf{s})]$ be the distribution of strings induced by the distribution of output token sequences d, and $\tilde{d}_{\texttt{dec}} = P_{\mathsf{s}\sim \tilde{d}(\mathsf{s})}[\texttt{dec}(\mathsf{s})]$ be the distribution of strings induced by the distribution of output token sequences $\tilde{d}$. Then, we have that

$$
\begin{aligned}
&\mathrm{KL}(p_{\texttt{dec}}, \tilde{d}_{\texttt{dec}}) = \\
&\quad = \sum_{\boldsymbol{\sigma}\in\Sigma^+} p_{\texttt{dec}}(\texttt{dec}(\mathsf{s}) = \boldsymbol{\sigma}) \ln\left(\frac{p_{\texttt{dec}}(\texttt{dec}(\mathsf{s}) = \boldsymbol{\sigma})}{\tilde{d}_{\texttt{dec}}(\texttt{dec}(\mathsf{s}) = \boldsymbol{\sigma})}\right) \\
&\quad = \sum_{\texttt{enc}(\boldsymbol{\sigma}),\boldsymbol{\sigma}\in\Sigma^+} p(\mathsf{s} = \texttt{enc}(\boldsymbol{\sigma})) \ln\left(\frac{p(\mathsf{s} = \texttt{enc}(\boldsymbol{\sigma}))}{\tilde{d}(\mathsf{s} = \texttt{enc}(\boldsymbol{\sigma}))}\right) \\
&\quad = \sum_{\mathbf{s}\in V^+:\, \mathbf{s} \text{ is canonical}} p(\mathsf{s} = \mathbf{s}) \ln\left(\frac{p(\mathsf{s} = \mathbf{s})}{\tilde{d}(\mathsf{s} = \mathbf{s})}\right) \\
&\quad = \mathrm{KL}(p, \tilde{d})
\end{aligned}
$$

because there is a one-to-one mapping determined by the encoder `enc` from any string to a canonical token sequence, and only canonical token sequences have positive probability under p and $\tilde{d}$. In contrast, one cannot claim the same for $\mathrm{KL}(p_{\texttt{dec}}, d_{\texttt{dec}})$ and $\mathrm{KL}(p, d)$, as the same string can have multiple tokenizations that have positive probability under d. Thus, we cannot conclude that $\mathrm{KL}(p_{\texttt{dec}}, \tilde{d}_{\texttt{dec}}) < \mathrm{KL}(p_{\texttt{dec}}, d_{\texttt{dec}})$.

Next, given this theoretical gap and since users typically derive value from the string that the output token sequence represents rather than the token sequence itself, we empirically compare the performance and efficiency of canonical and standard generation on the same three tasks from Section 3 (*i.e.*, translation, spell checking, and rephrasing), as well as a standard benchmark for multilingual LLM evaluation (*i.e.*, MGSM), using (open-weights) `Llama` and `Qwen` models.[12] To this end, we first sample 100 pairs of outputs per input prompt using standard and canonical generation with the same source of randomness, following Corvelo Benz et al. (2026). Then, we identify the pairs in which the output generated using standard generation are non-canonical, which are the only ones in which standard and canonical generation differ (under Gumbel-Max based sampling), and measure performance and time per token by (re-)sampling 10 continuations from the token in which the output became non-canonical under standard generation using both standard and canonical generation on each corresponding output (again with the same source of randomness).

To measure performance, we use (i) a quality score provided by a pre-trained neural network for reference-free translation evaluation (Guerreiro et al., 2024) in the translation task, (ii) the (normalized) Levenshtein edit distance (Levenshtein, 1966) between the generated text and the original text without typos in the spell checking task, (iii) cosine similarity of sentence embeddings (Reimers and Gurevych, 2019) of the original and rephrased text in the rephrasing task, and (iv) average accuracy in the MGSM task. Refer to Appendix F for additional details regarding the experimental setup.

Table 1 summarizes the results for the German language, which show that both canonical and standard generation are comparable both in terms of performance and efficiency. Here, the slightly lower performance

---

[12] In our experiments, for standard generation, we sample from the next-token distribution $d_{\mathbf{s}}(t)$ using the default categorical sampler in `PyTorch`, which is an implementation of Gumbel-Max sampling.

of canonical generation can be attributed to a limitation shared by constrained generation in general, namely, occasionally restricting the sampling space to low probability generation paths (Vieira et al., 2025b). Refer to Appendix G.2 for qualitatively similar results in other languages.

5 Discussion and Future Work

In this section, we highlight several limitations of our work and discuss avenues for future research.

Tasks and languages. Our experiments provides strong empirical evidence that tokenization multiplicity can occur on three natural language tasks, particularly in non-english languages. However, it would be interesting to study tokenization multiplicity on additional tasks. Moreover, it would be interesting to investigate whether commonly used practices to improve multilingual language generation, such as fine-tuning on different languages, using a different tokenizer per language, or using specialized models trained on mostly non-English text, may reduce the prevalence of tokenization multiplicity.

Methodology. Our main theoretical result (Theorem 2) reveals that, for BPE-, Unigram- and Wordpiece-based tokenizers, subsequences of canonical token sequences must also be canonical. It would be very interesting to better understand what property a tokenizer needs to satisfy for our result to hold. In this context, it would also be interesting to define relaxed notions of canonical tokenization applicable to stochastic tokenizers (Kudo, 2018; Provilkov et al., 2020), and adapt our theoretical result to this type of tokenizers.

Further, under canonical generation, we canonicalize the next-token distribution by redistributing the probability mass of tokens leading to non-canonical token sequences among the remaining tokens proportionally to their original probability mass. We have shown that, in comparison with the original next-token distribution, this particular canonicalized next-token distribution leads to a distribution of output sequences that is closer to the true distribution of token sequences. However, we have found that, in practice, canonical generation has slightly lower performance than standard generation. In future work, it would be worth to investigate global strategies beyond (next-token) sampling to redistribute the probability mass of non-canonical output token sequences to achieve better practical performance.

6 Conclusions

We have presented empirical evidence that, particularly for non-english outputs, both proprietary and open-weights LLMs often generate the same (output) string with different tokenizations, even under the same input prompt, and this multiplicity of tokenizations in turn leads to arbitrary, undesirable price variation. To address the problem of tokenization multiplicity, we have proposed canonical generation, a type of constrained generation that restricts LLMs to only generate the canonical tokenization of any output string, and introduced a simple and efficient sampling algorithm based on the Gumbel-Max trick to implement it. Further, we have shown that, in comparison with standard generation, the distribution of token sequences generated using canonical generation is provably closer to the true distribution of token sequences used during training, and the performance and runtime of LLMs using either method are comparable.

Acknowledgements. Gomez-Rodriguez acknowledges support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 945719). Tsirtsis acknowledges support from the Alexander von Humboldt Foundation in the framework of the Alexander von Humboldt Professorship (Humboldt Professor of Technology and Regulation awarded to Sandra Wachter) endowed by the Federal Ministry of Education and Research via the Hasso Plattner Institute.

References

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, Santa Clara, CA, April 2024. USENIX Association. ISBN 978-1-939133-39-7.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-Box Tuning for Language-Model-as-a-Service. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20841–20855. PMLR, 2022.

Emanuele La Malfa, Aleksandar Petrov, Simon Frieder, Christoph Weinhuber, Ryan Burnell, Raza Nazar, Anthony Cohn, Nigel Shadbolt, and Michael Wooldridge. Language-Models-as-a-Service: Overview of a New Paradigm and its Challenges. *J. Artif. Int. Res.*, 80, September 2024. ISSN 1076-9757.

S. Pais, J. Cordeiro, and M. L. Jamil. NLP-based platform as a service: a brief review. *Journal of Big Data*, 9(54), 2022.

Vasiliki Liagkou, Evangelia Filiopoulou, George Fragiadakis, Mara Nikolaidou, and Christos Michalakelis. The cost perspective of adopting Large Language Model-as-a-Service. In *2024 IEEE International Conference on Joint Cloud Computing (JCC)*, pages 80–83, 2024.

Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2023.

Jared Fernandez, Clara Na, Vashisth Tiwari, Yonatan Bisk, Sasha Luccioni, and Emma Strubell. Energy Considerations of Large Language Model Inference and Efficiency Optimizations. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32556–32569, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1563.

Renato Geh, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van Den Broeck. Where is the signal in tokenization space? In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3966–3979, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.230.

David D. Palmer. Tokenisation and Sentence Segmentation. In *Handbook of Natural Language Processing*, chapter 2, pages 24–25. Marcel Dekker, 2000.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models.* 3rd edition, 2025. Online manuscript released January 12, 2025.

Juan Luis Gastaldi, John Terilla, Luca Malagutti, Brian DuSell, Tim Vieira, and Ryan Cotterell. The Foundations of Tokenization: Statistical and Computational Concerns. In *The Thirteenth International Conference on Learning Representations*, 2025.

Buu Phan, Marton Havasi, Matthew Muckley, and Karen Ullrich. Understanding and Mitigating Tokenization Bias in Language Models. *arXiv preprint arXiv:2406.16829*, 2024.

Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. Toward a Theory of Tokenization in LLMs. *arXiv preprint arXiv:2404.08335*, 2025.

Martin Berglund and Brink van der Merwe. Formalizing BPE Tokenization. *Electronic Proceedings in Theoretical Computer Science*, 388:16–27, September 2023. ISSN 2075-2180. doi: 10.4204/eptcs.388.4.

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, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.38.

László Kozma and Johannes Voderholzer. Theoretical Analysis of Byte-Pair Encoding. *arXiv preprint arXiv:2411.08671*, 2024.

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, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7413–7425, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.459.

Ben Athiwaratkun, Shiqi Wang, Mingyue Shang, Yuchen Tian, Zijian Wang, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Robert Kwiatkowski, Ramesh Nallapati, Parminder Bhatia, and Bing Xiang. Token Alignment via Character Matching for Subword Completion. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15725–15738, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.929.

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, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 39–49, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-srw.5.

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

Aaditya K. Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier LLMs. *arXiv preprint arXiv:2402.14903*, 2024.

Siyang Liu, Naihao Deng, Sahand Sabour, Yilin Jia, Minlie Huang, and Rada Mihalcea. Task-Adaptive Tokenization: Enhancing Long-Form Text Generation Efficacy in Mental Health and Beyond. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15264–15281, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.944.

Eugene Kharitonov, Marco Baroni, and Dieuwke Hupkes. How BPE Affects Memorization in Transformers. *arXiv preprint arXiv:2110.02782*, 2021.

Aleksandar Petrov, Emanuele La Malfa, Philip H.S. Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

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, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.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, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–12, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.1.

Tim Vieira, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell. From Language Models over Tokens to Language Models over Characters. In *Forty-second International Conference on Machine Learning*, 2025a.

Mario Giulianelli, Luca Malagutti, Juan Luis Gastaldi, Brian DuSell, Tim Vieira, and Ryan Cotterell. On the Proper Treatment of Tokenization in Psycholinguistics. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18556–18572, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1032.

Renato Geh, Zilei Shao, and Guy Van Den Broeck. Adversarial Tokenization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 20738–20765, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1012.

Brian Siyuan Zheng, Alisa Liu, Orevaoghene Ahia, Jonathan Hayase, Yejin Choi, and Noah A. Smith. Broken Tokens? Your Language Model can Secretly Handle Non-Canonical Tokenizations. *arXiv preprint arXiv:2506.19004*, 2025.

Nikola Jovanović, Ismail Labiad, Tomáš Souček, Martin Vechev, and Pierre Fernandez. Watermarking Autoregressive Image Generation. *arXiv preprint arXiv:2506.16349*, 2025.

Guidance. Token Healing. `https://guidance.readthedocs.io/en/latest/example_notebooks/tutorials/token_healing.html`, 2023. Accessed: 2025-08-25.

Tim Vieira, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian DuSell, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J. O'Donnell, and Ryan Cotterell. Language Models over Canonical Byte-Pair Encodings. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b.

Shiyue Zhang, Vishrav Chaudhary, Naman Goyal, James Cross, Guillaume Wenzek, Mohit Bansal, and Francisco Guzman. How Robust is Neural Machine Translation to Language Imbalance in Multilingual Tokenizer Training? In Kevin Duh and Francisco Guzmán, editors, *Proceedings of the 15th biennial conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pages 97–116, Orlando, USA, September 2022. Association for Machine Translation in the Americas.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *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)*, pages 3118–3135, Online, August 2021. Association for Computational Linguistics.

Lingfeng Shen, Weiting Tan, Sihao Chen, Yunmo Chen, Jingyu Zhang, Haoran Xu, Boyuan Zheng, Philipp Koehn, and Daniel Khashabi. The Language Barrier: Dissecting Safety Challenges of LLMs in Multilingual Contexts. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2668–2680, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Wenxuan Wang, Zhaopeng Tu, Chang Chen, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael Lyu. All Languages Matter: On the Multilingual Safety of LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5865–5877, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. Multilingual Jailbreak Challenges in Large Language Models. In *The Twelfth International Conference on Learning Representations*, 2024.

Guoliang Dong, Haoyu Wang, Jun Sun, and Xinyu Wang. Evaluating and Mitigating Linguistic Discrimination in Large Language Models: Perspectives on Safety Equity and Knowledge Equity. In James Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 348–356. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/40. Main Track.

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore, December 2023. Association for Computational Linguistics.

Rafid Mahmood. Pricing and Competition for Generative AI. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Benjamin Laufer, Jon Kleinberg, and Hoda Heidari. Fine-tuning games: Bargaining and adaptation for general-purpose models. In *Proceedings of the ACM Web Conference 2024*, pages 66–76, 2024.

Will Cai, Tianneng Shi, Xuandong Zhao, and Dawn Song. Are You Getting What You Pay For? Auditing Model Substitution in LLM APIs. *arXiv preprint arXiv:2504.04715*, 2025.

Eden Saig, Ohad Einav, and Inbal Talgam-Cohen. Incentivizing quality text generation via statistical contracts. *Advances in Neural Information Processing Systems*, 37:51196–51222, 2024.

Dirk Bergemann, Alessandro Bonatti, and Alex Smolin. The Economics of Large Language Models: Token Allocation, Fine-Tuning, and Optimal Pricing. In *Proceedings of the 26th ACM Conference on Economics and Computation*, EC '25, page 786, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400719431. doi: 10.1145/3736252.3742625.

Guoheng Sun, Ziyao Wang, Bowei Tian, Meng Liu, Zheyu Shen, Shwai He, Yexiao He, Wanghao Ye, Yiting Wang, and Ang Li. CoIn: Counting the Invisible Reasoning Tokens in Commercial Opaque LLM APIs. *arXiv preprint arXiv:2505.13778*, 2025a.

Guoheng Sun, Ziyao Wang, Xuandong Zhao, Bowei Tian, Zheyu Shen, Yexiao He, Jinming Xing, and Ang Li. Invisible Tokens, Visible Bills: The Urgent Need to Audit Hidden Operations in Opaque LLM Services. *arXiv preprint arXiv:2505.18471*, 2025b.

Ander Artola Velasco, Stratis Tsirtsis, Nastaran Okati, and Manuel Gomez-Rodriguez. Is Your LLM Overcharging You? Tokenization, Transparency, and Incentives. *arXiv preprint arXiv:2505.21627*, 2025.

Ander Artola Velasco, Stratis Tsirtsis, and Manuel Gomez-Rodriguez. Auditing Pay-Per-Token in Large Language Models. In *The 29th International Conference on Artificial Intelligence and Statistics*, 2026.

Philip Gage. A New Algorithm for Data Compression. *C Users J.*, 12(2):23–38, February 1994. ISSN 0898-9788.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162.

Taku Kudo. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007.

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, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2089–2103, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.160.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report, 2024. *URL https://arxiv. org/abs/2407.10671*, 7:8, 2024.

Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG Van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE transactions on pattern analysis and machine intelligence*, 45(2):1353–1371, 2022.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The Curious Case of Neural Text Degeneration. In *International Conference on Learning Representations*, 2020.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. *Advances in neural information processing systems*, 27, 2014.

Jonathan Hayase, Alisa Liu, Noah A. Smith, and Sewoong Oh. Sampling from Your Language Model One Byte at a Time. *arXiv preprint arXiv:2506.14123*, 2025.

Nina Corvelo Benz, Stratis Tsirtsis, Eleni Straitouri, Ivi Chatzi, Ander Artola Velasco, Suhas Thejaswi, and Manuel Gomez-Rodriguez. Evaluation of Large Language Models via Coupled Token Generation. In *The 29th International Conference on Artificial Intelligence and Statistics*, 2026.

Nuno M. Guerreiro, Ricardo Rei, Daan van Stigt, Luisa Coheur, Pierre Colombo, and André F. T. Martins. xCOMET: Transparent Machine Translation Evaluation through Fine-grained Error Detection. *Transactions of the Association for Computational Linguistics*, 12:979–995, 2024. doi: 10.1162/tacl_a_00683.

V Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady*, 1966.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-Dropout: Simple and Effective Subword Regularization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.170.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967. doi: 10.1109/TIT.1967.1054010.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*, 2023.

A Tokenization Algorithms

There exists many tokenization algorithms to construct the set of tokens V, the encoder $\texttt{enc}$, and the decoder $\texttt{dec}$ characterizing a tokenizer $\mathcal{T}$. In the following, we review three popular tokenization algorithms, BPE (Gage, 1994; Sennrich et al., 2016), Unigram (Kudo, 2018) and Wordpiece (Song et al., 2021). We also discuss pretokenization, a preprocessing technique used to partition larger bodies of text before tokenization.

A.1 The BPE tokenization algorithm

The BPE tokenization algorithm (Gage, 1994; Sennrich et al., 2016) is used by most, if not all, state-of-the-art LLMs. In a nutshell, the BPE algorithm aims to create a tokenizer $\mathcal{T}$ with a set of tokens V corresponding to character sequences that appear frequently in a training set of strings $\mathcal{C}$. To this end, it proceeds as follows.

In an initialization phase, the algorithm sets i) Σ to be the set of all characters that appear at least once in $\mathcal{C}$, ii) V to be the set of single-character tokens, that is, for each $c \in \Sigma$, there exists one and only one $t \in V$ such that $\texttt{dec}(t) = c$, and iii) $\mathcal{S}$ to be the set of single-character token sequences $\mathbf{s} \in V^+$ representing all strings in $\mathcal{C}$. After the initialization phase, the algorithm proceeds iteratively for a predetermined number of iterations. At each iteration, it looks for the pair of tokens $t, t' \in V$ whose concatenation $t \shortmid t'$ appears most frequently in the set of token sequences $\mathcal{S}$, it creates a new token $t \circ t'$, where the symbol $\circ$ denotes the merge operation and $\texttt{dec}(t \circ t') = \texttt{dec}(t) \shortmid \texttt{dec}(t')$, and it adds the newly created token to V. Then, for each token sequence $\mathbf{s} \in \mathcal{S}$, it replaces all occurrences of $t \shortmid t'$ by $t \circ t'$ one by one. Lastly, it creates a merge rule $r_{t,t'}$, which specifies the replacement of $t \shortmid t'$ with $t \circ t'$, and adds it to an ordered sequence of merge rules $\mathcal{R}$.

After termination, the algorithm defines the encoder $\texttt{enc}$ and decoder $\texttt{dec}$ as follows. For any given token sequence $\mathbf{s} \in V^+$, $\texttt{dec}(\mathbf{s})$ transforms the sequence to a string one token at a time, in order, using the token definitions. For any given string $\boldsymbol{\sigma} \in \Sigma^+$, $\texttt{enc}(\boldsymbol{\sigma})$ first transforms the string to a sequence of single-character tokens. Then, it merges consecutive tokens from this sequence following the merge rules from $\mathcal{R}$, in order, until no merge rule is applicable, and it returns the resulting sequence—the canonical sequence.[13]

A.2 The Wordpiece tokenization algorithm

The Wordpiece algorithm is similar to BPE, in the sense that it builds the token vocabulary by iteratively merging tokens. However, the initialization phase, the merging criterion and the encoding function differ.

In the initialization phase, Σ is set to contain all characters that appear at least once in the training set of strings $\mathcal{C}$. Then, for each character $c \in \Sigma$ that appears at least once in $\mathcal{C}$, a single-character token t is added to V such that $\texttt{dec}(t) = c$, and $\mathcal{S}$ is initialized as a set that contains all single-character token sequences $\mathbf{s} \in V^+$ that represent all strings in $\mathcal{C}$. Interestingly, Wordpiece transforms characters (and substrings) inside words differently than characters (and substrings) at the beginning of words. Specifically, tokens representing characters (and substrings) inside words have a special prefix.

To build the vocabulary, Wordpiece proceeds iteratively by merging existing tokens and adding them to V until it reaches a predetermined size, similarly to BPE. However, the criterion to select which pair of tokens to merge is different. If $freq(\mathbf{s}')$ denotes the number of times that sequence $\mathbf{s}' \in V^+$ appears (as a subsequence) in the set of sequences $\mathcal{S}$, Wordpiece looks for the pair of tokens $t, t' \in V$ that maximizes the value of $\frac{freq(t \shortmid t')}{freq(t) \cdot freq(t')}$. Then, a new token $t \circ t'$ is added to V, where $\texttt{dec}(t \circ t') = \texttt{dec}(t) \shortmid \texttt{dec}(t')$, and all occurences of $t \shortmid t'$ in each token sequence $\mathbf{s} \in \mathcal{S}$ are replaced by $t \circ t'$. With this criterion, Wordpiece prefers to merge tokens whose concatenation appears commonly in $\mathcal{S}$, but they are not common individually.

After the above iterative process terminates, the algorithm defines the encoder and decoder functions as follows. For any token sequence $\mathbf{s} = t_1 \shortmid \cdots \shortmid t_n \in V^+$ with $n \in \mathbb{N}$, the decoder returns $\texttt{dec}(\mathbf{s}) = \texttt{dec}(t_1) \shortmid \cdots \shortmid \texttt{dec}(t_n)$ using the token definitions. Any string $\boldsymbol{\sigma} = c_1 \shortmid \cdots \shortmid c_m \in \Sigma^+$, $m \in \mathbb{N}$ given to the encoder is tokenized greedily from left to right, each time selecting the token in the vocabulary that represents the most characters starting from the beginning of the string. Specifically, the first token in $\texttt{enc}(\boldsymbol{\sigma})$ is the token $t \in V$ such that $\texttt{dec}(t) = c_1 \shortmid \cdots \shortmid c_i$, with $i \leq m$, and $\nexists t' \in V$ such that $\texttt{dec}(t') = c_1 \shortmid \cdots \shortmid c_j$ with $i < j \leq m$. In the above selection, if c_1 is inside a word, then t must contain the special prefix. This process continues in the same manner with the remaining string $c_{i+1} \shortmid \cdots \shortmid c_m$.

---

[13] If $t \shortmid t'$ appears multiple times in a token sequence, the merge rule $r_{t,t'}$ is applied in order of appearance in the sequence.

A.3 The Unigram tokenization algorithm

The Unigram algorithm aims to create a tokenizer $\mathcal{T}$ with a set of tokens V in order to minimize a loss when tokenizing a training set of strings $\mathcal{C}$. In the initialization phase, Σ is set to contain all the characters that appear at least once in $\mathcal{C}$. Unlike the BPE algorithm, which iteratively adds tokens to the vocabulary V, Unigram starts with a large vocabulary and removes tokens from it until it reaches a predetermined size. This initial large vocabulary can be set in multiple ways, such as applying the BPE algorithm on $\mathcal{C}$ with many iterations, or initializing it with tokens that decode to the most frequently occuring substrings in $\mathcal{C}$.

After the initial vocabulary has been set, the algorithm proceeds in iterations, each time computing a loss over the strings in $\mathcal{C}$ and the current vocabulary, and removing a batch of tokens from the vocabulary (typically 10% or 20% of tokens) whose removal minimizes this loss. In each iteration, every token t in the current vocabulary V is assigned a probability score $r(t) = \frac{freq(t)}{\sum_{t' \in V} freq(t')}$, where $freq(t)$ denotes the number of times that the token t appears in all possible tokenizations of the strings in $\mathcal{C}$. For each token $t \in V$, the loss over the training set is computed as $\sum_{\boldsymbol{\sigma} \in \mathcal{C}} - \log(r_{V \setminus \{t\}}(\boldsymbol{\sigma}))$, where $r_V(\boldsymbol{\sigma}) = \max_{\mathbf{s} \in V^+, \texttt{dec}(\mathbf{s}) = \boldsymbol{\sigma}} r(\mathbf{s})$ denotes the probability score of the most likely tokenization of $\boldsymbol{\sigma}$ under vocabulary V, and the probability score of tokenization $\mathbf{s} = t_1 \mid \cdots \mid t_n$, with $n \in \mathbb{N}$, is simply $r(\mathbf{s}) = r(t_1) \ldots r(t_n)$. The tokens that minimize this loss are removed from the vocabulary and the process repeats until the vocabulary reaches a predetermined size.

After the vocabulary has been finalized, the encoder is set to tokenize a string $\boldsymbol{\sigma} \in \Sigma^+$ by finding its most likely tokenization under the final vocabulary V, *i.e.*, $\texttt{enc}(\boldsymbol{\sigma}) = \arg\max_{\mathbf{s} \in V^+, \texttt{dec}(\mathbf{s}) = \boldsymbol{\sigma}} r(\mathbf{s})$, using the Viterbi algorithm (Viterbi, 1967), and the decoder decodes all tokens in V the same way as in the original, large vocabulary.

B Proof of Theorem 2

In this section, we prove Theorem 2 by showing individually that each tokenization algorithm—BPE, Wordpiece, and Unigram—builds tokenizers which are non-recovering.

B.1 BPE-based tokenizers are non-recovering

In order to show that BPE-based tokenizers are non-recovering, we define some additional notation regarding the BPE tokenization algorithm.

When tokenizing a string $\boldsymbol{\sigma} = c_1 \shortmid \cdots \shortmid c_n$, with $c_i \in \Sigma$ and $n \in \mathbb{N}$, according to the BPE algorithm, we use the term *merge* and write $m = (r_{t,t'}, i, j)$ to refer to a single application of merge rule $r_{t,t'} \in \mathcal{R}$ on two consecutive tokens $t \shortmid t'$ that correspond to the substring of characters $c_i \shortmid \cdots \shortmid c_j$ in $\boldsymbol{\sigma}$. To tokenize $\boldsymbol{\sigma}$, merges are performed following a unique merge sequence $M = (m_1, \ldots, m_{|M|})$, where the merges are ordered $m_1 \prec \cdots \prec m_{|M|}$, first by the order in which the merge rule they refer to appears in $\mathcal{R}$, and second by position of merged token pairs in the sequence. The notation $m \prec m'$, for $m = (r, i, j), m' = (r', i', j')$ with $r, r' \in \mathcal{R},\ i, j, i', j' \in [n]$, means that either r appears before r' in $\mathcal{R}$, or $r = r'$ and $i < i'$.

We now define an operator that, applied to a merge sequence M that tokenizes the string $\boldsymbol{\sigma}$, specifies the subsequence of merges that are applied to a certain substring of $\boldsymbol{\sigma}$. Further, we define *shift equivalence*, referring to merge sequences whose merges correspond to the exact same merge rule sequence applied to different positions in a string (shifted by a constant).

Definition 4. *Let $\mathbf{s} = t_1 \shortmid \cdots \shortmid t_{|\mathbf{s}|} \in V^+$ be a tokenization of $\boldsymbol{\sigma} = c_1 \shortmid \cdots \shortmid c_{|\boldsymbol{\sigma}|} \in \Sigma^+$ obtained by applying merge sequence $M = (m_1, \ldots, m_n)$. For any continuous token subsequence $\mathbf{s}'$ of $\mathbf{s}$ spanning the substring $\boldsymbol{\sigma}' = c_u \shortmid \cdots \shortmid c_v$, $1 \leq u < v \leq |\boldsymbol{\sigma}|$, the operator $[M]_{\mathbf{s}'}$ denotes the subsequence of merges in M such that $m = (r, i, j) \in [M]_{\mathbf{s}'}$ if $m \in M$ and $u \leq i < j \leq v$.*

Definition 5. *Two merge sequences $M = (m_1, \ldots, m_{|M|})$, $M' = (m'_1, \ldots, m'_{|M'|})$ are* shift equivalent*, denoted by $M \overset{\rightarrow}{\equiv} M'$, if $|M| = |M'|$ and there exists $n \in \mathbb{Z}$ such that for all $i \in \{1, \ldots, |M|\}$ with $m_i = (r, j, k)$, $r \in \mathcal{R}$, $k > j > 0$, it holds that $m'_i = (r, j + n, k + n)$.*

Before we prove that BPE-based tokenizers are non-recovering, we show that the merge sequence that creates the tokenization $\mathbf{s} = \mathbf{s}_1 \shortmid \cdots \shortmid \mathbf{s}_n$ from a string $\boldsymbol{\sigma}$, can be partitioned into n disjoint (non-continuous) subsequences of merges, that create the tokenizations $\mathbf{s}_1, \ldots, \mathbf{s}_n$ from the corresponding substring of $\boldsymbol{\sigma}$.

Lemma 1. *Let $\mathbf{s} \in V^+$ be a tokenization of $\boldsymbol{\sigma} = c_1 \shortmid \cdots \shortmid c_{|\boldsymbol{\sigma}|} \in \Sigma^+$ obtained by applying merge sequence $M_{\mathbf{s}}$. For any partition $\mathbf{s} = \mathbf{s}_1 \shortmid \cdots \shortmid \mathbf{s}_n$, where $\mathbf{s}_i \in V^+, i \in [n], n \in \mathbb{N}$, the following hold:*

1. *For each $\mathbf{s}_i \in \mathbf{s}$, there exists a merge sequence $M_{\mathbf{s}_i}$ such that applying $M_{\mathbf{s}_i}$ to the string $\texttt{dec}(\mathbf{s}_i)$ creates $\mathbf{s}_i$ and $[M_{\mathbf{s}}]_{\mathbf{s}_i} \overset{\rightarrow}{\equiv} M_{\mathbf{s}_i}$.*

2. *For all $\mathbf{s}_i, \mathbf{s}_j \in \mathbf{s}, i \neq j$, if $m \in [M_{\mathbf{s}}]_{\mathbf{s}_i}$ then $m \notin [M_{\mathbf{s}}]_{\mathbf{s}_j}$ and vice-versa,*

3. *For each merge $m \in M_{\mathbf{s}}$ there exists $\mathbf{s}_i \in \mathbf{s}$ such that $m \in [M_{\mathbf{s}}]_{\mathbf{s}_i}$.*

Proof. 1. If $\mathbf{s}_i$ is a tokenization of a single character $\texttt{dec}(\mathbf{s}_i) = c$, then $[M_{\mathbf{s}}]_{\mathbf{s}_i}$ is the empty sequence and the statement holds trivially. Assume $\mathbf{s}_i$ is a tokenization of the substring $c_u \shortmid \cdots \shortmid c_v$ of $\boldsymbol{\sigma}$, with $v > u > 0$, and $[M_{\mathbf{s}}]_{\mathbf{s}_i} = (m_1, \ldots, m_n)$. By Definition 4, all merges $m = (r, j_1, j_2) \in M_{\mathbf{s}}$ with $u \leq j_1 < j_2 \leq v$ belong in $[M_{\mathbf{s}}]_{\mathbf{s}_i}$, so these merges tokenize $c_u \shortmid \cdots \shortmid c_v$ into $\mathbf{s}_i$. Then, the merge sequence $M_{\mathbf{s}_i} = (m'_1, \ldots, m'_n)$, where for all $k \in [n]$ it holds that $m_k = (r, j_1, j_2)$ and $m'_k = (r, j_1 - u, j_2 - u)$, $r \in \mathcal{R}$, contains the same merge rules in the same order, but with indices shifted left by u. So if $M_{\mathbf{s}_i}$ is applied to the string $\texttt{dec}(\mathbf{s}_i)$ it will create $\mathbf{s}_i$.

2. If $[M_{\mathbf{s}}]_{\mathbf{s}_i}$ or $[M_{\mathbf{s}}]_{\mathbf{s}_j}$ are the empty sequence, meaning $\mathbf{s}_i$ or $\mathbf{s}_j$ are a tokenization of only a single character in $\boldsymbol{\sigma}$, then the statement holds trivially. If $\mathbf{s}_i$ is a tokenization of the substring $c_u \shortmid \cdots \shortmid c_v$ and $\mathbf{s}_j$ is a tokenization of the substring $c_{u'} \shortmid \cdots \shortmid c_{v'}$, since $i \neq j$ it must be that either $u < v < u' < v'$

or $u' < v' < u < v$. But for all $m = (r, i_1, i_2) \in [M_{\mathbf{s}}]_{\mathbf{s}_i}$ it holds that $u \leq i_1 < i_2 \leq v$, and for all $m' = (r', j_1, j_2) \in [M_{\mathbf{s}}]_{\mathbf{s}_j}$ it holds that $u' \leq j_1 < j_2 \leq v'$. Intuitively, it is not possible for a merge to span two different subsequences $\mathbf{s}_i, \mathbf{s}_j$ in the partition of $\mathbf{s}$, because then (part of) $\mathbf{s}_i$ and $\mathbf{s}_j$ would be merged.

3. Each merge $m = (r, j, k) \in M_{\mathbf{s}}, r \in \mathcal{R}$ must have $1 \leq j < k \leq |\boldsymbol{\sigma}|$. Because the whole string $\boldsymbol{\sigma}$ is tokenized into $\mathbf{s}$ and, by definition, the token merged by m cannot be part of two different subsequences in the partition, there must exist $\mathbf{s}_i \in \mathbf{s}$ that is a tokenization of a substring $c_u \shortmid \cdots \shortmid c_v$ of $\boldsymbol{\sigma}$ with $u \leq j < k \leq v$. So by Definition 4, $m \in [M_{\mathbf{s}}]_{\mathbf{s}_i}$.

□

Building up on Lemma 1, we show that BPE-based tokenizers are non-recovering.

Lemma 2. *BPE-based tokenizers are non-recovering.*

Proof. Assume that $\mathbf{s} \shortmid t$ is canonical. Then, there exists a unique merge sequence $M_{\mathbf{s} \shortmid t}$ that creates it following the BPE algorithm. From Lemma 1, $M_{\mathbf{s} \shortmid t}$ can be split into $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ and $[M_{\mathbf{s} \shortmid t}]_t$, where $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ contains the merges that create $\mathbf{s}$ and $[M_{\mathbf{s} \shortmid t}]_t$ contains the merges that create t. From Lemma 1, there exists a merge sequence $M_{\mathbf{s}}$ that creates $\mathbf{s}$ when applied to $\mathtt{dec}(\mathbf{s})$ and $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}} \overset{\rightrightarrows}{\equiv} M_{\mathbf{s}}$. Because $\mathbf{s}$ is a prefix of $\mathbf{s} \shortmid t$, the index shift is zero and we have that $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}} = M_{\mathbf{s}}$.

Since $\mathbf{s}$ tokenized according to $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}} = (m_1, \ldots, m_n)$ is non-canonical, there must exist a different, canonical tokenization $\mathbf{s}' \neq \mathbf{s}$ of the same character string, $\mathtt{dec}(\mathbf{s}) = \mathtt{dec}(\mathbf{s}')$. Let $M_{\mathbf{s}'} = (m'_1, \ldots, m'_{n'})$ be the unique merge sequence that creates $\mathbf{s}'$ from $\mathtt{dec}(\mathbf{s})$ according to the BPE algorithm. Because $M_{\mathbf{s}'} \neq [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$, it must be that either there exists at least one i, $i \leq \min(n, n')$, such that $m_i \neq m'_i$, or $m_i = m'_i$ for all $i \in [\min(n, n')]$ but $n \neq n'$.

We will first examine the first case. Let $m_i \in [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ and $m'_i \in M_{\mathbf{s}'}$ be the first merges that are different between $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ and $M_{\mathbf{s}'}$, meaning $\forall j < i: \; m_j = m'_j$, for $m_j \in [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$, $m'_j \in M_{\mathbf{s}'}$. Because $\mathbf{s}'$ is canonical and $\mathbf{s}$ is not, it must be that $m'_i \prec m_i$. We will now compare $M_{\mathbf{s} \shortmid t}$, $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ and $M_{\mathbf{s}'}$. There are two sub-cases:

1. The first i merges in $M_{\mathbf{s} \shortmid t}$ are the same as in $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$. This means that the first $i-1$ merges are the same as in $M_{\mathbf{s}'}$. Then, merge m_i being applied instead of $m'_i \prec m_i$ on substring $\mathtt{dec}(\mathbf{s})$, implies that $M_{\mathbf{s} \shortmid t}$ cannot be the merge sequence that creates the canonical tokenization of $\mathtt{dec}(\mathbf{s} \shortmid t)$ according to BPE.

2. The first i merges in $M_{\mathbf{s} \shortmid t}$ are not the same as in $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$. This means that there exists at least one merge $m \in M_{\mathbf{s} \shortmid t}$ among the first i merges in $M_{\mathbf{s} \shortmid t}$ such that $m \notin [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$. For any such merge m, as $m \notin [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$, it must hold that $m \in [M_{\mathbf{s} \shortmid t}]_{\boldsymbol{t}}$ by Lemma 1. So, in $M_{\mathbf{s} \shortmid t}$, merge m_i is preceded by the first $i-1$ merges of $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ and merge m. By Lemma 1, m does not affect the tokens that will create $\mathbf{s}$, so the only merges in $M_{\mathbf{s} \shortmid t}$ before m_i that affect $\mathbf{s}$ are the first $i-1$ merges of $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$, which are the same as $M_{\mathbf{s}'}$. Then, as in case 1, merge m_i being applied instead of $m'_i \prec m_i$, implies that $M_{\mathbf{s} \shortmid t}$ cannot be the merge sequence that creates the canonical tokenization of $\mathtt{dec}(\mathbf{s} \shortmid t)$ according to BPE.

We will now examine the case where $m_i = m'_i$ for all $i \in [\min(n, n')]$, $m_i \in [M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$, $m'_i \in M_{\mathbf{s}'}$ but $n \neq n'$. If $n > n'$, then there exists at least one merge that can be applied on $\mathbf{s}'$ after all merges of $M_{\mathbf{s}'}$ are done, which means that $\mathbf{s}'$ cannot be canonical. If $n' > n$, then there exists at least one merge that can be applied on $\mathbf{s}$ after all merges of $[M_{\mathbf{s} \shortmid t}]_{\mathbf{s}}$ are done. This merge can also be applied on $\mathbf{s} \shortmid t$, which means that $\mathbf{s} \shortmid t$ cannot be canonical.

All cases lead to a contradiction, which implies that $\mathbf{s} \shortmid t$ is non-canonical. We have shown that if $\mathbf{s}$ is non-canonical then $\mathbf{s} \shortmid t$ is also non-canonical. Thus, BPE-based tokenizers are non-recovering. □

B.2 Unigram-based tokenizers are non-recovering.

Theorem 6. *Unigram-based tokenizers are non-recovering.*

Proof. If $\mathbf{s}$ is non-canonical according to Unigram, then let $\mathbf{s}'$ denote the canonical tokenization of the same character string, $\texttt{dec}(\mathbf{s}) = \texttt{dec}(\mathbf{s}')$. Because $\mathbf{s}'$ is canonical, it must be that $r(\mathbf{s}') > r(\mathbf{s})$. It follows that $r(\mathbf{s} \mid t) = r(\mathbf{s})r(t) < r(\mathbf{s}')r(t) = r(\mathbf{s}' \mid t)$, so $\mathbf{s} \mid t$ cannot be the canonical tokenization of $\texttt{dec}(\mathbf{s} \mid t)$. We have shown that if $\mathbf{s}$ is non-canonical then $\mathbf{s} \mid t$ is also non-canonical. Thus, Unigram-based tokenizers are non-recovering. ☐

B.3 Wordpiece-based tokenizers are non-recovering.

Theorem 7. *Wordpiece-based tokenizers are non-recovering.*

Proof. If $\mathbf{s} = t_1 \mid \cdots \mid t_n$ is non-canonical according to Wordpiece, then let $\mathbf{s}' = t'_1 \mid \cdots \mid t'_{n'}$ denote the canonical tokenization of the same character string, $\texttt{dec}(\mathbf{s}) = \texttt{dec}(\mathbf{s}')$, $n, n' \in \mathbb{N}$. Because $\mathbf{s}' \neq \mathbf{s}$, there must exist at least one $i \leq \min(n, n')$ such that $t_i \neq t'_i$. It is impossible that $t_i = t'_i$ for all $i \in \min(n, n')$ but $n \neq n'$, because then $\texttt{dec}(\mathbf{s}) \neq \texttt{dec}(\mathbf{s}')$, as one would be a prefix of the other. Let t_i, t'_i, with $i \in \min(n, n')$ be the first different token between $\mathbf{s}$ and $\mathbf{s}'$, *i.e.*, $\forall j < i : t_j = t'_j$ but $t_i \neq t'_i$. Since $\mathbf{s}'$ is canonical, it must be that $|t'_i| > |t_i|$, where $|t| = |\texttt{dec}(t)|$ represents the size of token t based on how many characters in Σ it encodes. Because $\mathbf{s}$ is a prefix of $\mathbf{s} \mid t$, the first i tokens are the same, but $\mathbf{s} \mid t$ cannot be canonical because at (token) index i there exists $t'_i \in V$ that encodes more characters than t_i, $|t'_i| > |t_i|$. We have shown that if $\mathbf{s}$ is non-canonical then $\mathbf{s} \mid t$ is also non-canonical. Thus, Wordpiece-based tokenizers are non-recovering. ☐

C Non-recoverability under Pretokenization

State-of-the-art LLMs use a tool called *pretokenizer* in order to split long strings into segments that can be tokenized simultaneously and independent of each other. Formally, the pretokenizer is a function $\mathtt{pre} : \Sigma^+ \to (\Sigma^+)^+$, where $(\Sigma^+)^+$ represents sequences of strings in Σ^+, such that for string $\boldsymbol{\sigma} \in \Sigma^+$, $\mathtt{pre}(\boldsymbol{\sigma}) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n)$ where $\boldsymbol{\sigma} = \boldsymbol{\sigma}_1 \shortmid \cdots \shortmid \boldsymbol{\sigma}_n$, $n \in \mathbb{N}$. The encoder with pretokenizer can then be defined as $\mathtt{enc}_{\mathtt{pre}}(\boldsymbol{\sigma}) = \mathtt{enc}(\boldsymbol{\sigma}_1) \shortmid \cdots \shortmid \mathtt{enc}(\boldsymbol{\sigma}_n)$, where $\mathtt{enc} : \Sigma^+ \to V^+$ is an encoder based on the tokenization algorithm used in conjunction with the pretokenizer. We extend the definition of canonical sequences to account for the effect of the pretokenizer.

Definition 8. *Let* $\mathcal{T} = (\Sigma, V, \mathtt{enc}_{\mathtt{pre}}, \mathtt{dec})$ *be a tokenizer and* $\boldsymbol{\sigma} \in \Sigma^+$. *A tokenization* $\mathbf{s} \in V^+$, $\mathtt{dec}(\mathbf{s}) = \boldsymbol{\sigma}$ *of* $\boldsymbol{\sigma}$ *is* canonical *if* $\mathbf{s} = \mathtt{enc}_{\mathtt{pre}}(\boldsymbol{\sigma})$*, where* $\mathtt{enc}(\boldsymbol{\sigma}) = \mathtt{enc}(\boldsymbol{\sigma}_1) \shortmid \cdots \shortmid \mathtt{enc}(\boldsymbol{\sigma}_n)$ *and* $\mathtt{pre}(\boldsymbol{\sigma}) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n)$, $n \in \mathbb{N}$.

Pretokenizers typically work by greedily matching prefixes of a string to a regular expression, splitting when the prefix stops matching, and continuing with the remaining suffix. If string $\boldsymbol{\sigma}$ is a match, then $\mathtt{pre}(\boldsymbol{\sigma}) = \boldsymbol{\sigma}$, and if $\mathtt{pre}(\boldsymbol{\sigma}) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n), n \in \mathbb{N}, \boldsymbol{\sigma} \in \Sigma^+$ then $\mathtt{pre}(\boldsymbol{\sigma}_i) = \boldsymbol{\sigma}_i$ for all $i \in [n]$. Regular expressions used by pretokenizers additionally satisfy a property called *closed under prefix* (Hayase et al., 2025), though some exceptions apply related to handling whitespace and common english contractions.

Definition 9. *A pretokenizer* $\mathtt{pre}$ *is* closed under prefix *if for any string* $\boldsymbol{\sigma} \in \Sigma^+$ *where* $\mathtt{pre}(\boldsymbol{\sigma}) = \boldsymbol{\sigma}$ *and any prefix* $\boldsymbol{\sigma}'$ *of* $\boldsymbol{\sigma}$*, it holds that* $\mathtt{pre}(\boldsymbol{\sigma}') = \boldsymbol{\sigma}'$.

In words, any prefix of a string that is a match to the regular expression is also a match, or equivalently, if a string is not a match then no superstring of it is a match.

We show that tokenizers with pretokenization remain non-recovering when the pretokenizer is closed under prefix.

Theorem 10. *Let* $\mathcal{T} = (\Sigma, V, \mathtt{enc}, \mathtt{dec})$ *be a non-recovering tokenizer and let* $\mathtt{pre}$ *be a pretokenizer closed under prefix. Then, tokenizer* $\mathcal{T}' = (\Sigma, V, \mathtt{enc}_{\mathtt{pre}}, \mathtt{dec})$ *is non-recovering.*

Proof. Let $\boldsymbol{\sigma} = \boldsymbol{\sigma}_1 \shortmid \cdots \shortmid \boldsymbol{\sigma}_n = \mathtt{dec}(\mathbf{s})$, where $\mathtt{pre}(\boldsymbol{\sigma}) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n)$, $\boldsymbol{\sigma}_i = \mathtt{dec}(\mathbf{s}_i)$ for all $i \in [n]$ and $\mathbf{s} = \mathbf{s}_1 \shortmid \cdots \shortmid \mathbf{s}_n$, $n \in \mathbb{N}$. Additionally, let $\boldsymbol{\sigma}_t = \mathtt{dec}(t)$, so $\boldsymbol{\sigma} \shortmid \boldsymbol{\sigma}_t = \mathtt{dec}(\mathbf{s} \shortmid t)$. By definition 9 and because t is a single token, it must hold that either $\mathtt{pre}(\boldsymbol{\sigma} \shortmid \boldsymbol{\sigma}_t) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n, \boldsymbol{\sigma}_t)$ or $\mathtt{pre}(\boldsymbol{\sigma} \shortmid \boldsymbol{\sigma}_t) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n \shortmid \boldsymbol{\sigma}_t)$. For each $\boldsymbol{\sigma}_i$, it holds by definition that $\mathtt{pre}(\boldsymbol{\sigma}_i) = \boldsymbol{\sigma}_i$, so $\mathtt{enc}_{\mathtt{pre}}(\boldsymbol{\sigma}_i) = \mathtt{enc}(\mathtt{dec}(\mathbf{s}_i))$.

By definition 8, for $\mathbf{s}$ to be non-canonical there must be at least one $\mathbf{s}_i$ where $\mathbf{s}_i \neq \mathtt{enc}_{\mathtt{pre}}(\mathtt{dec}(\mathbf{s}_i)) = \mathtt{enc}(\mathtt{dec}(\mathbf{s}_i))$, so $\mathbf{s}_i$ is non-canonical. As each substring $\boldsymbol{\sigma}_i$ is tokenized independently, if $i < n$, or if $i = n$ and $\mathtt{pre}(\boldsymbol{\sigma} \shortmid \boldsymbol{\sigma}_t) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n, \boldsymbol{\sigma}_t)$, then $\mathbf{s}_i$ is also part of $\mathbf{s} \shortmid t$, so $\mathbf{s} \shortmid t$ is non-canonical. Alternatively, if $i = n$ and $\mathtt{pre}(\boldsymbol{\sigma} \shortmid \boldsymbol{\sigma}_t) = (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n \shortmid \boldsymbol{\sigma}_t)$, then because $\mathcal{T}$ is non-recovering, $\mathbf{s}_n$ being non-canonical implies that $\mathbf{s}_n \shortmid t$ is also non-canonical, therefore $\mathbf{s} \shortmid t$ is non-canonical. In all cases, it holds that if $\mathbf{s}$ is non-canonical then $\mathbf{s} \shortmid t$ is also non-canonical. Thus, tokenizer $\mathcal{T}'$ is non-recovering. $\square$

D Proof of Theorem 3

Here, we provide the proof of Theorem 3, which we restate below.

Theorem 3. Let d be absolutely continuous with respect to p. Moreover, assume that there exist $\mathbf{s} \in V^+$ and $t_1, t_2 \in V$ such that $\mathbf{s} \shortmid t_1$ is non-canonical with $d(\mathbf{s} \shortmid t_1) > 0$ and $\mathbf{s} \shortmid t_2$ is canonical with $p(\mathbf{s} \shortmid t_2) > 0$ and $d(\mathbf{s} \shortmid t_2) > 0$. Then, it holds that

$$\mathrm{KL}(p, \tilde{d}) < \mathrm{KL}(p, d). \tag{3}$$

Proof. Assume there exists $\hat{\mathbf{s}} \in V^+$, $t_1, t_2 \in V$ such that $\hat{\mathbf{s}} \shortmid t_1$ is non-canonical and $d(\hat{\mathbf{s}} \shortmid t_1) > 0$ and $\hat{\mathbf{s}} \shortmid t_2$ is canonical and $p(\hat{\mathbf{s}} \shortmid t_2) > 0$ and $d(\hat{\mathbf{s}} \shortmid t_2) > 0$. Given any token sequence $\mathbf{s} \in V^+$, let $p_{\mathbf{s}} = P[T | \boldsymbol{S} = \mathbf{s}]$ be the true next token distribution and $d_{\mathbf{s}}$, $\tilde{d}_{\mathbf{s}}$ be the next token distribution and canonicalized next token distribution given by the LLM. Then, $d_{\hat{\mathbf{s}}}(t_1) > 0$. Then, we have that $Z = \sum_{t \in V:\ \hat{\mathbf{s}} \shortmid t \text{ is canonical}} d_{\hat{\mathbf{s}}}(t) < 1$. By definition of $\tilde{d}_{\hat{\mathbf{s}}}$, this implies that for all $t \in V$ such that $\tilde{d}_{\hat{\mathbf{s}}}(t) > 0$, we have that

$$\frac{\tilde{d}_{\hat{\mathbf{s}}}(t)}{d_{\hat{\mathbf{s}}}(t)} > 1 \tag{4}$$

Note that, because $\hat{\mathbf{s}}$ is canonical (by Theorem 2 and because $p(\hat{\mathbf{s}} \shortmid t_2) > 0$) and $d(\hat{\mathbf{s}}) > 0$, it implies that $\hat{\mathbf{s}}$ also has positive probability under $\tilde{d}$, *i.e.*, $\tilde{d}(\hat{\mathbf{s}}) > 0$. In particular, by definition of $\tilde{d}$ we know that $\tilde{d}(\hat{\mathbf{s}})/d(\hat{\mathbf{s}}) \geq 1$ and thus using Eq. 4 it follows that for any t such that $\tilde{d}_{\hat{\mathbf{s}}}(t) > 0$, $\tilde{d}(\hat{\mathbf{s}} \shortmid t) > 0$ and

$$\frac{\tilde{d}(\hat{\mathbf{s}} \shortmid t)}{d(\hat{\mathbf{s}} \shortmid t)} = \frac{\tilde{d}(\hat{\mathbf{s}})}{d(\hat{\mathbf{s}})} \cdot \frac{\tilde{d}_{\hat{\mathbf{s}}}(t)}{d_{\hat{\mathbf{s}}}(t)} > 1 \tag{5}$$

We show that the difference in KL-divergence of p from d and p from $\tilde{d}$ is greater than zero. First, we rewrite the difference as follows:

$$\begin{aligned}
\mathrm{KL}(p, d) - \mathrm{KL}(p, \tilde{d}) &= \sum_{\mathbf{s} \in V^+} p(\mathbf{s}) \log\left(\frac{p(\mathbf{s})}{d(\mathbf{s})}\right) - \sum_{\mathbf{s} \in V^+} p(\mathbf{s}) \log\left(\frac{p(\mathbf{s})}{\tilde{d}(\mathbf{s})}\right) \\
&= \sum_{\mathbf{s} \in V^+} p(\mathbf{s}) \left[\log\left(\frac{p(\mathbf{s})}{d(\mathbf{s})}\right) - \log\left(\frac{p(\mathbf{s})}{\tilde{d}(\mathbf{s})}\right)\right] \\
&= \sum_{\mathbf{s} \in V^+} p(\mathbf{s}) \log\left(\frac{\tilde{d}(\mathbf{s})}{d(\mathbf{s})}\right) \qquad (6) \\
&= \sum_{\mathbf{s} \in V^+:\ \tilde{d}(\mathbf{s}) > 0} p(\mathbf{s}) \log\left(\frac{\tilde{d}(\mathbf{s})}{d(\mathbf{s})}\right) \qquad (7)
\end{aligned}$$

where the first equations follow from simple manipulations and Eq. 7 follows from the following argument. Whenever $\tilde{d}(\mathbf{s}) = 0$, it implies that either $d(\mathbf{s}) = 0$ or that $\mathbf{s}$ is non-canonical. Both cases imply that $p(\mathbf{s}) = 0$ (either by absolute continuity or non-canonicity). Whenever $p(\mathbf{s})$ and $\tilde{d}(\mathbf{s})$ are zero, the contribution of the corresponding term in Eq. 6 is interpreted as zero because $\lim_{x \to 0^+} x \log x = 0$.

We can break up Eq. 7 into two types of summand. For any $\mathbf{s} \neq \hat{\mathbf{s}} \shortmid t, t \in V$ and $\tilde{d}(\mathbf{s}) > 0$, it readily follows from the definition of $\tilde{d}_{\mathbf{s}}$ that

$$p(\mathbf{s}) \log\left(\frac{\tilde{d}(\mathbf{s})}{d(\mathbf{s})}\right) \geq p(\mathbf{s}) \log(1) = 0$$

For any $\mathbf{s} = \hat{\mathbf{s}} \shortmid t, t \in V$ and $\tilde{d}(\mathbf{s}) > 0$ and $p(\mathbf{s}) > 0$, it follows from Eq. 5 that

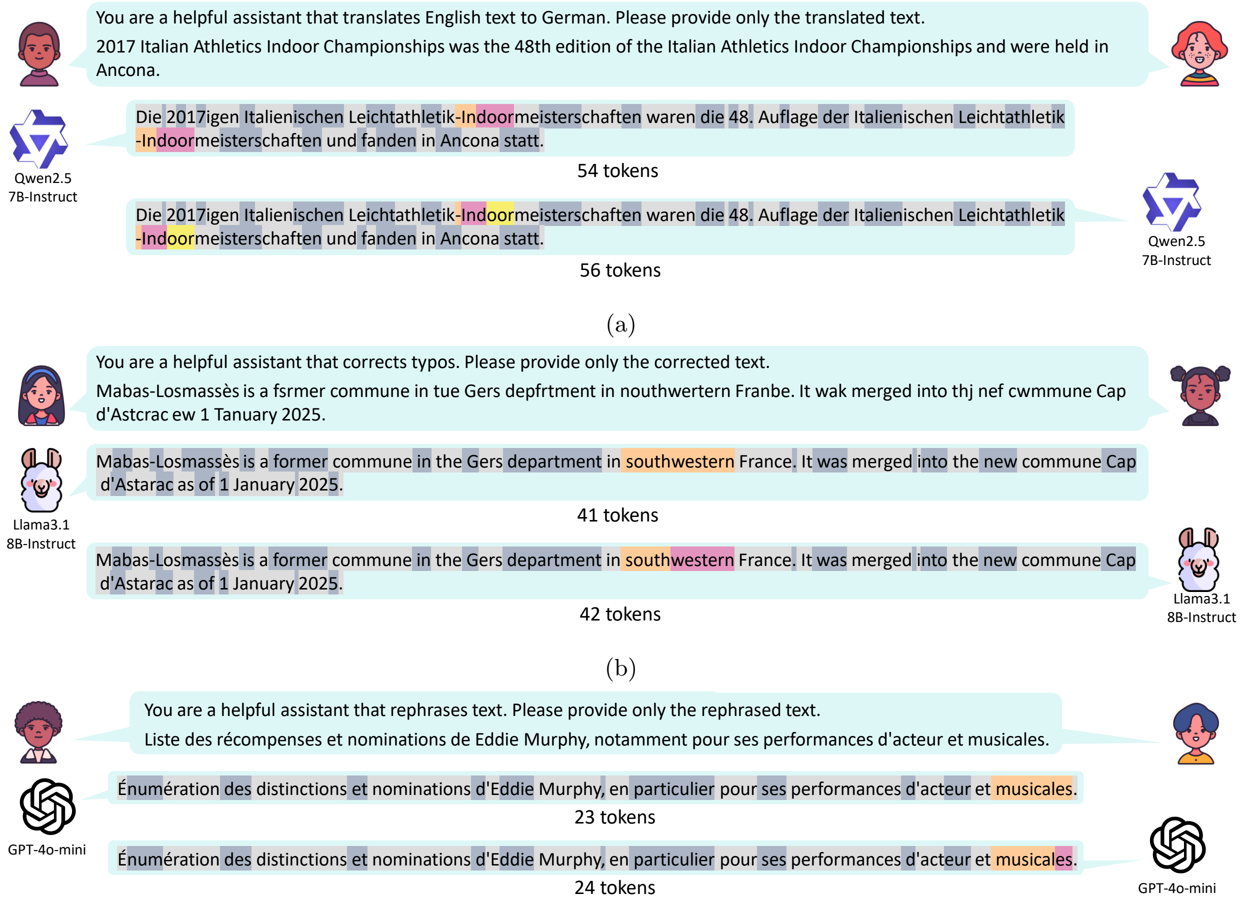
$$p(\mathbf{s}) \log\left(\frac{\tilde{d}(\mathbf{s})}{d(\mathbf{s})}\right) > p(\mathbf{s}) \log(1) = 0$$

Thus, we can conclude that, as there exist $t_2 \in V$ such that $p(\hat{\mathbf{s}} \shortmid t_2) > 0$ and $\tilde{d}(\hat{\mathbf{s}} \shortmid t_2) > 0$,

$$\mathrm{KL}(p, d) - \mathrm{KL}(p, \tilde{d}) > 0.$$

$\square$

E Examples of Tokenization Multiplicity



You are a helpful assistant that translates English text to German. Please provide only the translated text.

2017 Italian Athletics Indoor Championships was the 48th edition of the Italian Athletics Indoor Championships and were held in Ancona.

Die 2017igen Italienischen Leichtathletik-Indoormeisterschaften waren die 48. Auflage der Italienischen Leichtathletik-Indoormeisterschaften und fanden in Ancona statt.

54 tokens

Die 2017igen Italienischen Leichtathletik-Indoormeisterschaften waren die 48. Auflage der Italienischen Leichtathletik-Indoormeisterschaften und fanden in Ancona statt.

56 tokens

(a)

You are a helpful assistant that corrects typos. Please provide only the corrected text.

Mabas-Losmassès is a fsrmer commune in tue Gers depfrtment in nouthwertern Franbe. It wak merged into thj nef cwmmune Cap d'Astcrac ew 1 Tanuary 2025.

Mabas-Losmassès is a former commune in the Gers department in southwestern France. It was merged into the new commune Cap d'Astarac as of 1 January 2025.

41 tokens

Mabas-Losmassès is a former commune in the Gers department in southwestern France. It was merged into the new commune Cap d'Astarac as of 1 January 2025.

42 tokens

(b)

You are a helpful assistant that rephrases text. Please provide only the rephrased text.

Liste des récompenses et nominations de Eddie Murphy, notamment pour ses performances d'acteur et musicales.

Énumération des distinctions et nominations d'Eddie Murphy, en particulier pour ses performances d'acteur et musicales.

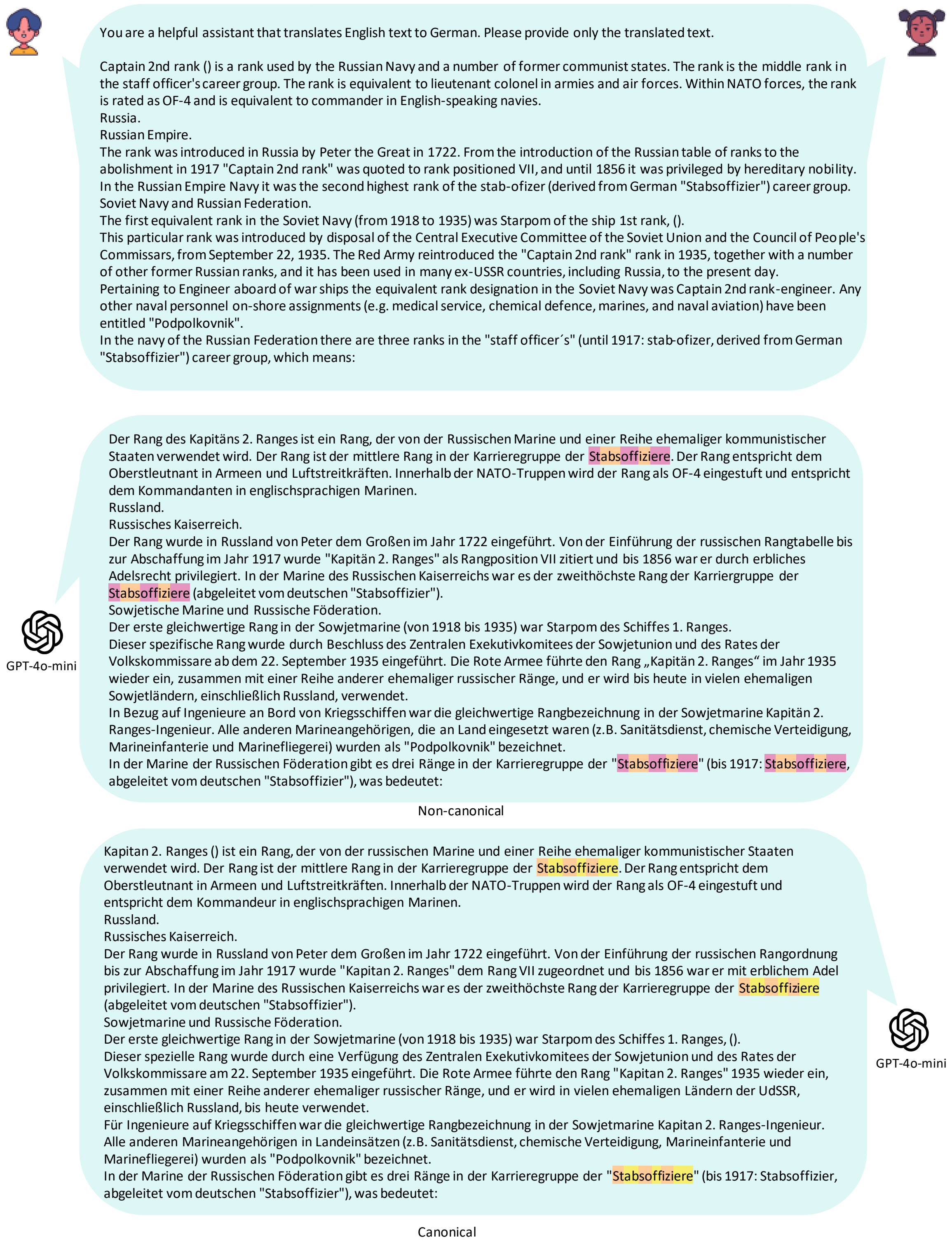
23 tokens

Énumération des distinctions et nominations d'Eddie Murphy, en particulier pour ses performances d'acteur et musicales.

24 tokens

(c)

Figure 5: **Examples of tokenization multiplicity in (a) translation, (b) spell checking, and (c) rephrasing.** In each example, the top box shows the input prompt, which consists of an instruction of the task and the accompanying Wikipedia text to be processed. The latter two boxes show two outputs generated by (a) `Qwen2.5-7B-Instruct`, (b) `Llama3.1-8B-Instruct` and (c) `GPT-4o-mini` as a response to the input prompt, corresponding to the same string but with two different tokenizations.



You are a helpful assistant that translates English text to German. Please provide only the translated text.

Captain 2nd rank () is a rank used by the Russian Navy and a number of former communist states. The rank is the middle rank in the staff officer's career group. The rank is equivalent to lieutenant colonel in armies and air forces. Within NATO forces, the rank is rated as OF-4 and is equivalent to commander in English-speaking navies.
Russia.
Russian Empire.
The rank was introduced in Russia by Peter the Great in 1722. From the introduction of the Russian table of ranks to the abolishment in 1917 "Captain 2nd rank" was quoted to rank positioned VII, and until 1856 it was privileged by hereditary nobility. In the Russian Empire Navy it was the second highest rank of the stab-ofizer (derived from German "Stabsoffizier") career group.
Soviet Navy and Russian Federation.
The first equivalent rank in the Soviet Navy (from 1918 to 1935) was Starpom of the ship 1st rank, ().
This particular rank was introduced by disposal of the Central Executive Committee of the Soviet Union and the Council of People's Commissars, from September 22, 1935. The Red Army reintroduced the "Captain 2nd rank" rank in 1935, together with a number of other former Russian ranks, and it has been used in many ex-USSR countries, including Russia, to the present day.
Pertaining to Engineer aboard of war ships the equivalent rank designation in the Soviet Navy was Captain 2nd rank-engineer. Any other naval personnel on-shore assignments (e.g. medical service, chemical defence, marines, and naval aviation) have been entitled "Podpolkovnik".
In the navy of the Russian Federation there are three ranks in the "staff officer´s" (until 1917: stab-ofizer, derived from German "Stabsoffizier") career group, which means:

Der Rang des Kapitäns 2. Ranges ist ein Rang, der von der Russischen Marine und einer Reihe ehemaliger kommunistischer Staaten verwendet wird. Der Rang ist der mittlere Rang in der Karrieregruppe der Stabsoffiziere. Der Rang entspricht dem Oberstleutnant in Armeen und Luftstreitkräften. Innerhalb der NATO-Truppen wird der Rang als OF-4 eingestuft und entspricht dem Kommandanten in englischsprachigen Marinen.
Russland.
Russisches Kaiserreich.
Der Rang wurde in Russland von Peter dem Großen im Jahr 1722 eingeführt. Von der Einführung der russischen Rangtabelle bis zur Abschaffung im Jahr 1917 wurde "Kapitän 2. Ranges" als Rangposition VII zitiert und bis 1856 war er durch erbliches Adelsrecht privilegiert. In der Marine des Russischen Kaiserreichs war es der zweithöchste Rang der Karrieregruppe der Stabsoffiziere (abgeleitet vom deutschen "Stabsoffizier").
Sowjetische Marine und Russische Föderation.
Der erste gleichwertige Rang in der Sowjetmarine (von 1918 bis 1935) war Starpom des Schiffes 1. Ranges.
Dieser spezifische Rang wurde durch Beschluss des Zentralen Exekutivkomitees der Sowjetunion und des Rates der Volkskommissare ab dem 22. September 1935 eingeführt. Die Rote Armee führte den Rang „Kapitän 2. Ranges" im Jahr 1935 wieder ein, zusammen mit einer Reihe anderer ehemaliger russischer Ränge, und er wird bis heute in vielen ehemaligen Sowjetländern, einschließlich Russland, verwendet.
In Bezug auf Ingenieure an Bord von Kriegsschiffen war die gleichwertige Rangbezeichnung in der Sowjetmarine Kapitän 2. Ranges-Ingenieur. Alle anderen Marineangehörigen, die an Land eingesetzt waren (z.B. Sanitätsdienst, chemische Verteidigung, Marineinfanterie und Marinefliegerei) wurden als "Podpolkovnik" bezeichnet.
In der Marine der Russischen Föderation gibt es drei Ränge in der Karrieregruppe der "Stabsoffiziere" (bis 1917: Stabsoffiziere, abgeleitet vom deutschen "Stabsoffizier"), was bedeutet:

Non-canonical

Kapitan 2. Ranges () ist ein Rang, der von der russischen Marine und einer Reihe ehemaliger kommunistischer Staaten verwendet wird. Der Rang ist der mittlere Rang in der Karrieregruppe der Stabsoffiziere. Der Rang entspricht dem Oberstleutnant in Armeen und Luftstreitkräften. Innerhalb der NATO-Truppen wird der Rang als OF-4 eingestuft und entspricht dem Kommandeur in englischsprachigen Marinen.
Russland.
Russisches Kaiserreich.
Der Rang wurde in Russland von Peter dem Großen im Jahr 1722 eingeführt. Von der Einführung der russischen Rangordnung bis zur Abschaffung im Jahr 1917 wurde "Kapitan 2. Ranges" dem Rang VII zugeordnet und bis 1856 war er mit erblichem Adel privilegiert. In der Marine des Russischen Kaiserreichs war es der zweithöchste Rang der Karrieregruppe der Stabsoffiziere (abgeleitet vom deutschen "Stabsoffizier").
Sowjetmarine und Russische Föderation.
Der erste gleichwertige Rang in der Sowjetmarine (von 1918 bis 1935) war Starpom des Schiffes 1. Ranges, ().
Dieser spezielle Rang wurde durch eine Verfügung des Zentralen Exekutivkomitees der Sowjetunion und des Rates der Volkskommissare am 22. September 1935 eingeführt. Die Rote Armee führte den Rang "Kapitan 2. Ranges" 1935 wieder ein, zusammen mit einer Reihe anderer ehemaliger russischer Ränge, und er wird in vielen ehemaligen Ländern der UdSSR, einschließlich Russland, bis heute verwendet.
Für Ingenieure auf Kriegsschiffen war die gleichwertige Rangbezeichnung in der Sowjetmarine Kapitan 2. Ranges-Ingenieur. Alle anderen Marineangehörigen in Landeinsätzen (z.B. Sanitätsdienst, chemische Verteidigung, Marineinfanterie und Marinefliegerei) wurden als "Podpolkovnik" bezeichnet.
In der Marine der Russischen Föderation gibt es drei Ränge in der Karrieregruppe der "Stabsoffiziere" (bis 1917: Stabsoffizier, abgeleitet vom deutschen "Stabsoffizier"), was bedeutet:

Canonical

Figure 6: **Example of tokenization multiplicity in long, similar outputs.** The top box consists of a translation instruction and the accompanying Wikipedia text to be translated. The latter two boxes show two outputs generated by `gpt-4o-mini` as response to the input prompt, corresponding to the similar strings but with two different tokenizations for the word "Stabsoffiziere".

F Additional Experimental Details

Hardware setup. Our experiments using open-weights models are executed on a compute server equipped with 2 × Intel Xeon Gold 5317 CPU, 1,024 GB main memory, and 2 × A100 Nvidia Tesla GPU (80 GB, Ampere Architecture). In each experiment a single Nvidia A100 GPU is used.

Datasets and languages. As input texts for our experiments on the translation, spell checking and rephrasing tasks, we used articles from the most recent Wikipedia dumps[14] as of December 3rd 2025 in different languages. See Table 2 for a full list of languages and the shortened names used in our plots in section 3 and Appendix G. We extracted plain text from the articles using the `wikiextractor` tool[15] and sampled 100 articles from each language with length between 30 and 300 characters. For the experiment with longer outputs at the end of section 3, we sampled 100 articles in the english language with length between 1000 and 3000 characters. For the spell checking task, we randomly replaced, with probability 10%, lowercase latin characters in the input with a random different lowercase latin character. Additionally, for the experiments on the MGSM task in section 4.3 we used the MGSM benchmark (Shi et al., 2023), consisting of 250 grade-school maths problems translated in different languages, but we only considered languages in latin script. The reason our experiments are solely focused on languages using the latin script is that for most non-latin scripts tokens very often encode at most one character, therefore many, if not all, output strings cannot be generated under multiple tokenizations.

Models and parameters. Table 3 lists the models used in our experiments, as well as the shortened names used in our results in sections 3, 4.3 and Appendix G. All inferences were performed with temperature set to 1.0. The system prompts used in the MGSM task were adopted from an open-source evaluation library[16], which uses a 0-shot chain-of-thought prompting technique, while the system prompts for the other tasks are shown in Tables 4, 5, and 6. For `gpt-5-mini`, we used the minimum reasoning setting and subtracted from the output token count any reasoning tokens that are not visible to the user.

API details. To investigate tokenization multiplicity in proprietary models, we used the publicly available official API services from OpenAI, Google, and Anthropic. Further, to measure the canonicity of an output, its tokenization must be disclosed by the API and a tokenizer must be publically available.[17] However, the API services for `gpt5m`, `gemini` and `claude` return only the output string and number of generated tokens, without disclosing the exact tokenization. For these models, we can identify some cases of non-canonicity, when the number of generated tokens does not match the number of tokens in the canonical tokenization.

Reproducibility. We have released all code and data required to reproduce our results at the following repository: https://github.com/Networks-Learning/Tokenization-Multiplicity. However, the exact outputs of the proprietary models are not always reproducible. Specifically, the API services for `gemini`, `claude` and `gpt5m` do not allow setting a random seed for deterministic outputs, while for `gpt4om` and `gpt4.1` setting a random seed is possible but the output is deterministic only if it is accompanied by the same `system_fingerprint` field, which cannot be controlled by the user. Therefore, we have included in the repository all outputs from these models where we observed tokenization multiplicity, and believe that one can obtain qualitatively similar results by running our code.

---

[14] https://dumps.wikimedia.org/

[15] https://github.com/attardi/wikiextractor

[16] https://github.com/openai/simple-evals/blob/main/mgsm_eval.py

[17] OpenAI provide a public tokenizer: https://github.com/openai/tiktoken

Full name	Shortened name
German	de
French	fr
Portuguese	pt
English	en
Turkish	tr
Swahili	sw

Table 2: Languages used in our experiments.

Full name	Shortened name
`Llama-3.1-8B-Instruct`	`Llama8B`
`Qwen2.5-7B-Instruct`	`Qwen7B`
`gpt-4o-mini`	`gpt4om`
`gpt-4.1`	`gpt4.1`
`gpt-5-mini`	`gpt5m`
`gemini-2.5-flash-lite`	`gemini`
`claude-3-haiku-20240307`	`claude`

Table 3: Models used in our experiments.

System: You are a helpful assistant that translates LANG-1 text to LANG-2. Please provide only the translated text.

Table 4: System prompt used for the translation task. LANG-1 and LANG-2 correspond to full names of languages from Table 2.

System: You are a helpful assistant that corrects typos. Please provide only the corrected text.

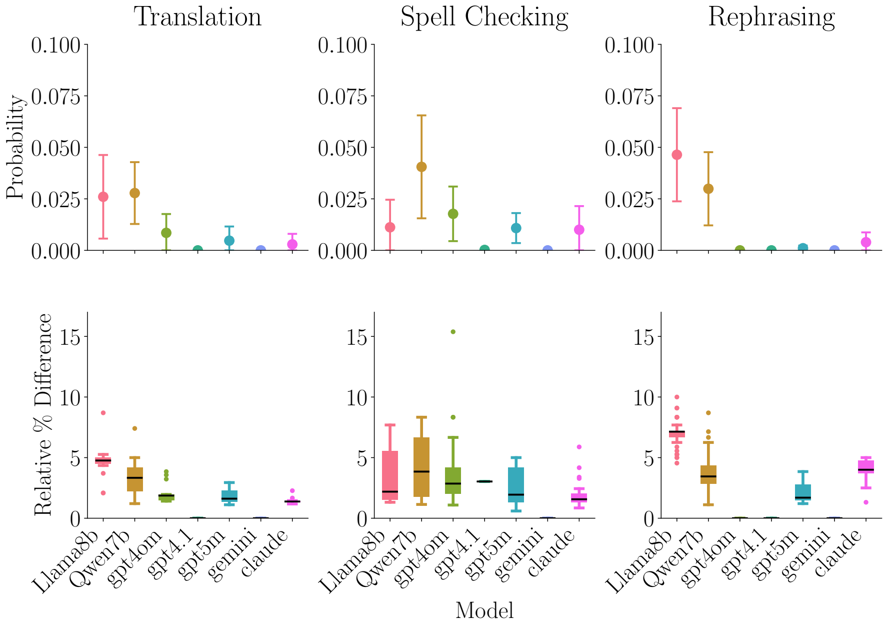
Table 5: System prompt used for the spell checking task.

System: You are a helpful assistant that rephrases text. Please provide only the rephrased text.

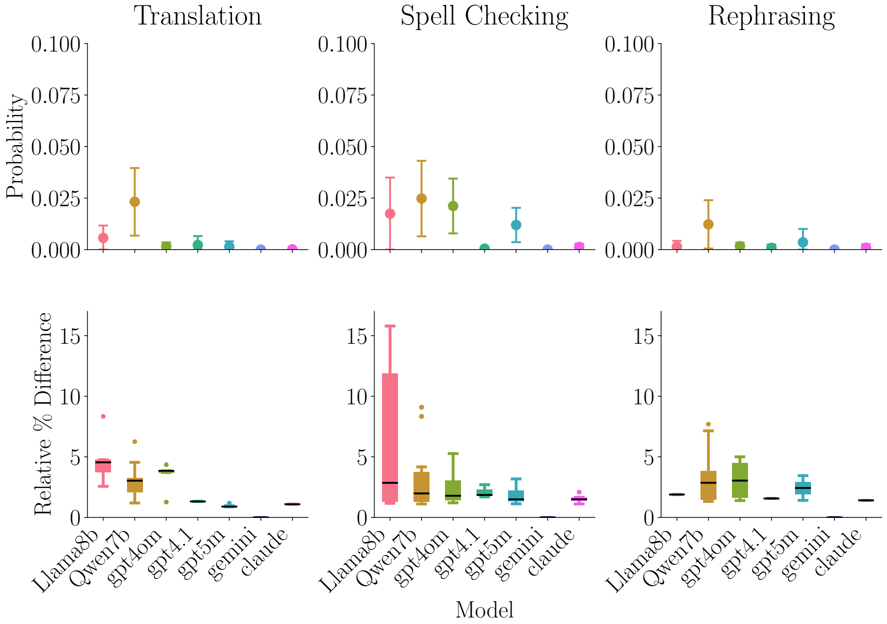
Table 6: System prompt used for the rephrasing task.

G Additional Experimental Results

G.1 Additional Experimental Results on Tokenization Multiplicity

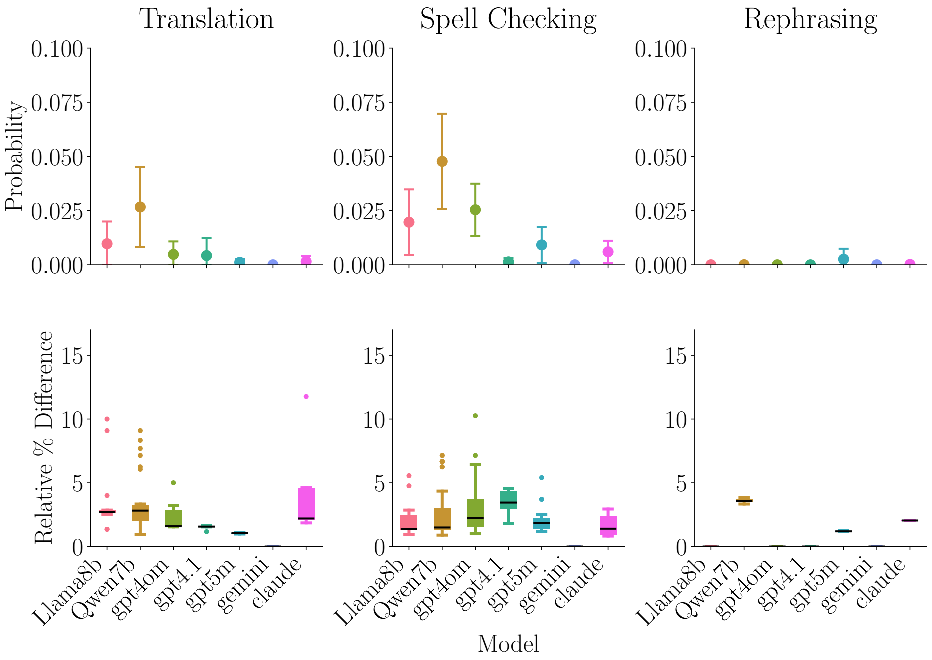


(a) German

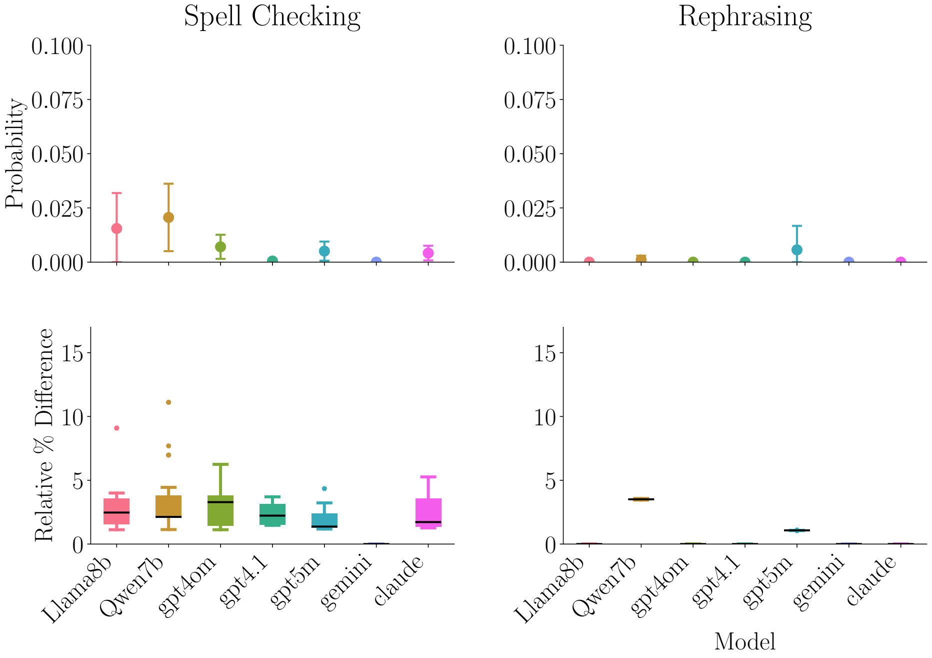


(b) French

Figure 7: **Probability of tokenization multiplicity and magnitude of price variation for tasks in the (a) German and (b) French languages.**

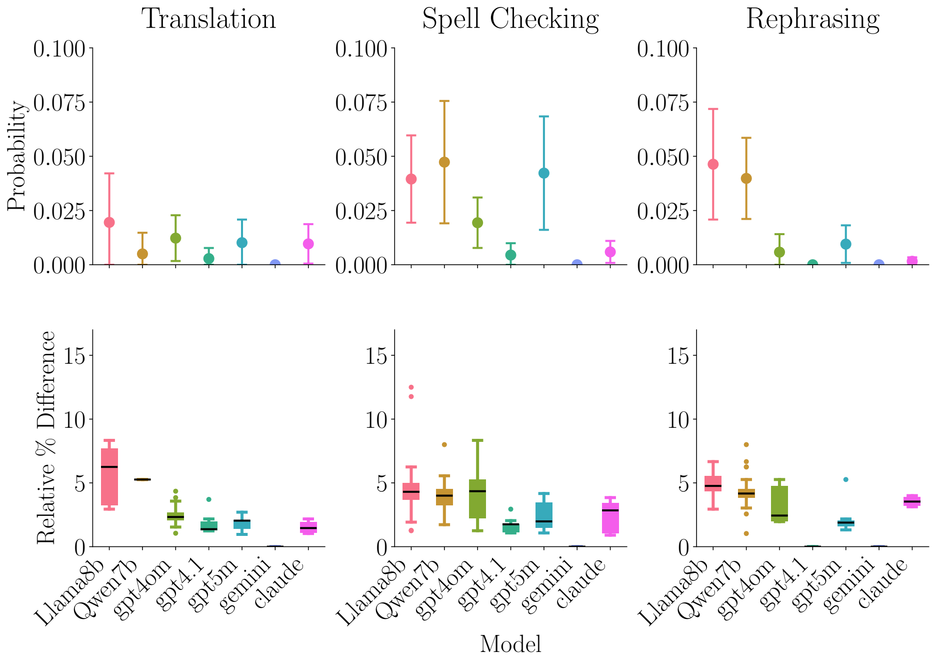


(a) Portuguese

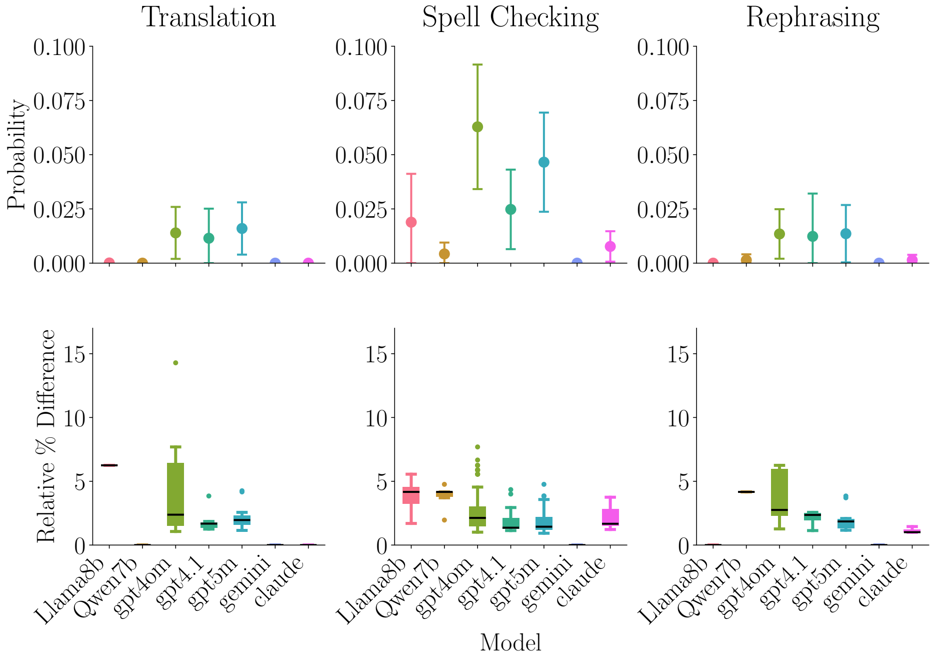


(b) English

Figure 8: **Probability of tokenization multiplicity and magnitude of price variation for tasks in the (a) Portuguese and (b) English languages.**



(a) Turkish



(b) Swahili

Figure 9: **Probability of tokenization multiplicity and magnitude of price variation for tasks in the (a) Turkish and (b) Swahili languages.**

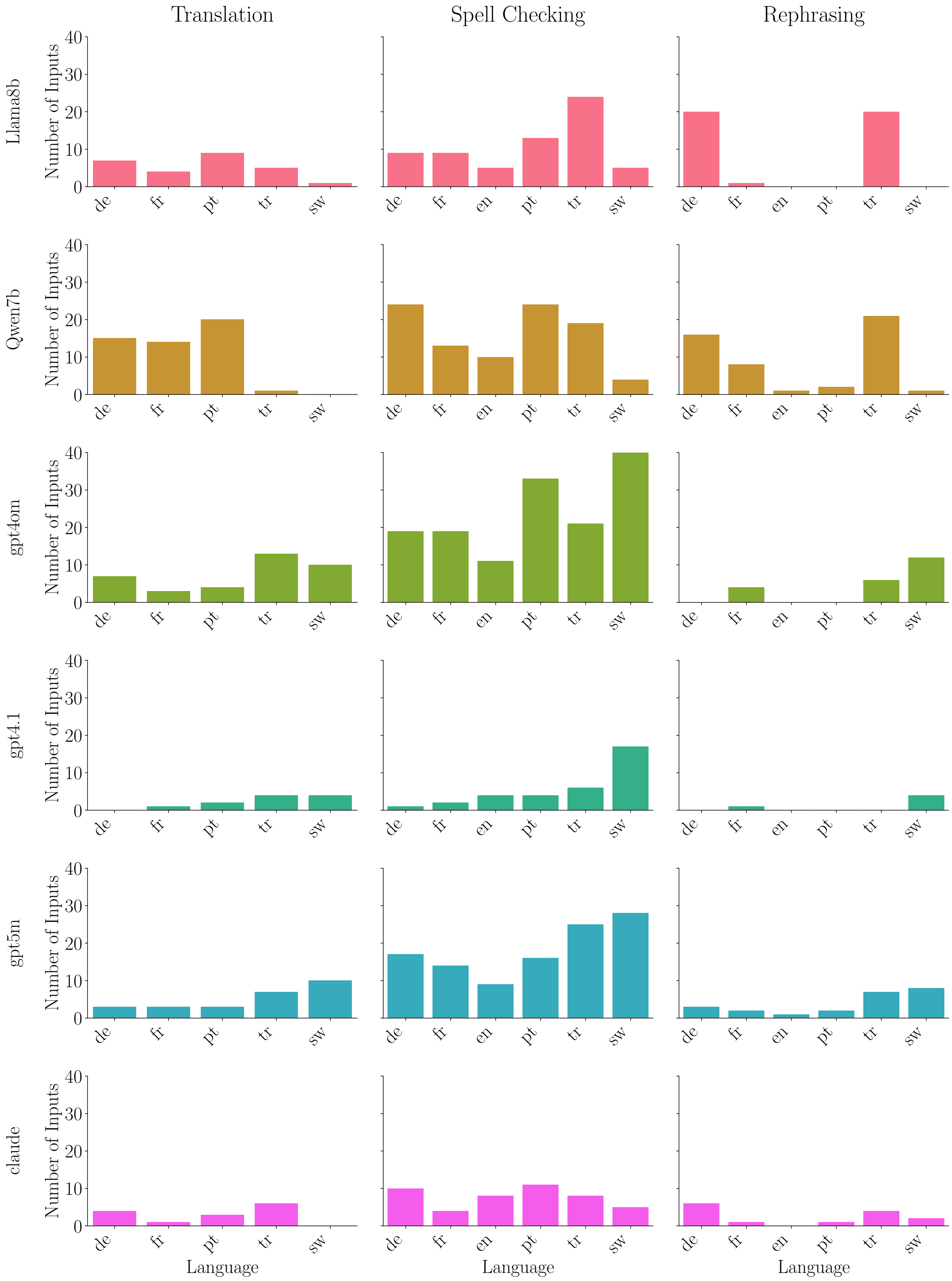


Figure 10: **Tokenization multiplicity across languages.** The plots show the number of inputs prompts for which we observe at least two outputs given by each LLM with the same string but different tokenization lengths. Each row corresponds to a different LLM, panels corresponds to one of the three tasks we consider in our experiments and pairs of letters on the x-axis correspond to different languages.

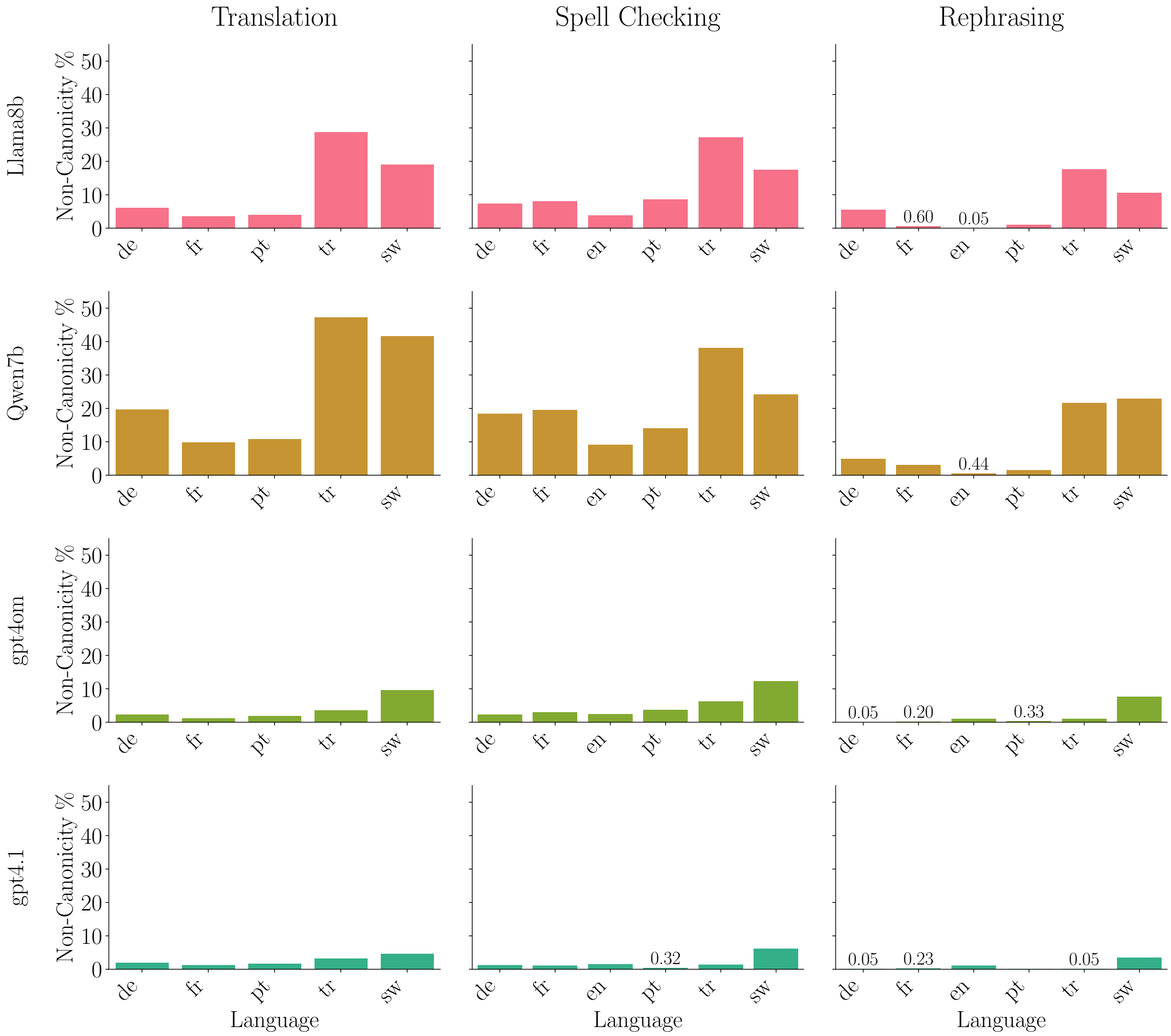


Figure 11: **Percentage of non-canonical outputs from LLMs that disclose the tokenization.** The plots show the percentage of outputs whose generated tokenization does not match the canonical tokenization of the same string. Each row corresponds to a different LLM, panels corresponds to one of the three tasks we consider in our experiments and pairs of letters on the x-axis correspond to different languages.

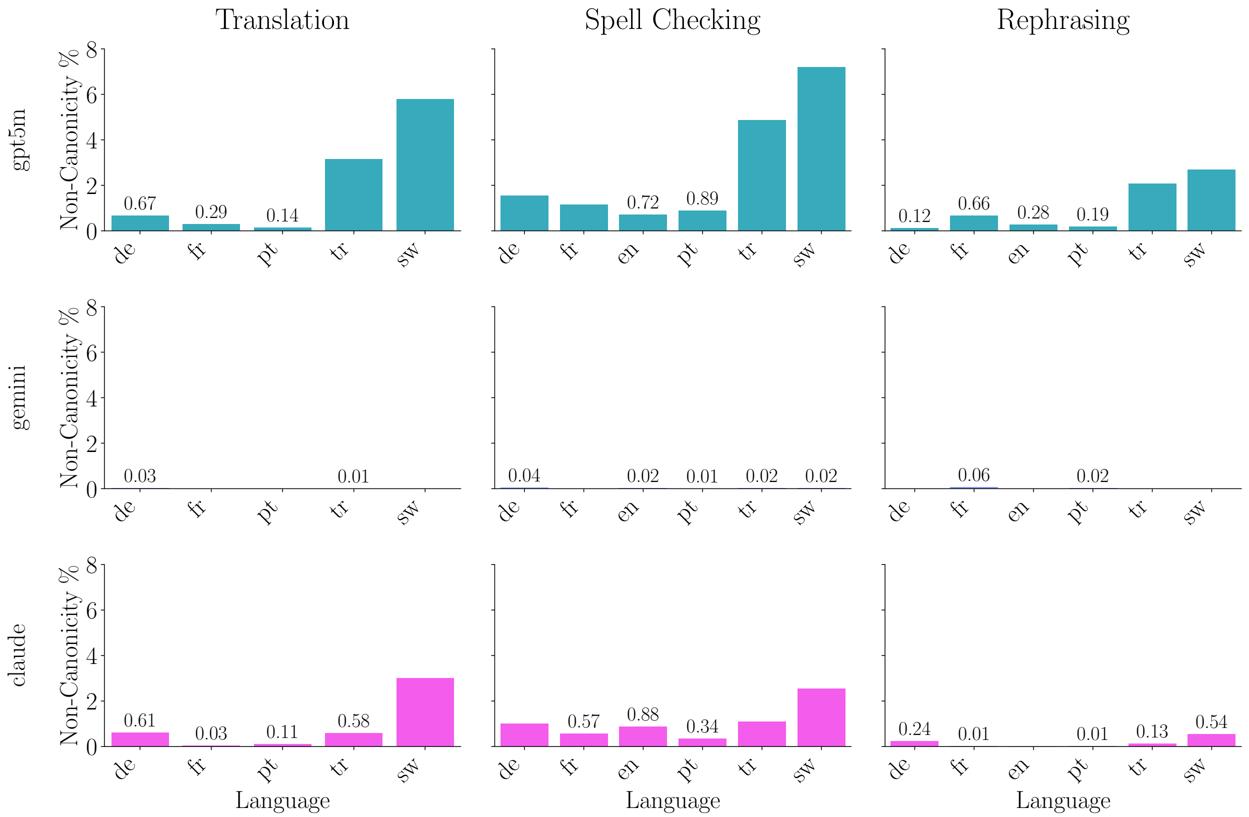


Figure 12: **Percentage of non-canonical outputs (lower bound) from LLMs that do not disclose the tokenization.** The plots show the percentage of outputs whose generated tokenization length does not match the canonical tokenization length for the same string. Each row corresponds to a different LLM, panels corresponds to one of the three tasks we consider in our experiments and pairs of letters on the x-axis correspond to different languages.

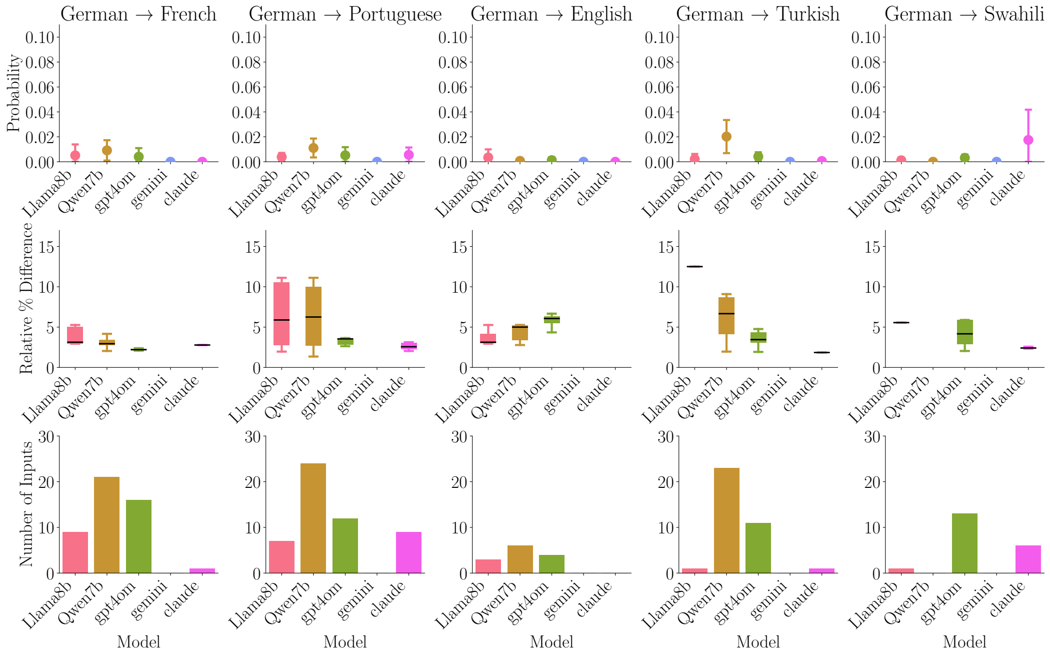


Figure 13: **Tokenization multiplicity in the translation task from German to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

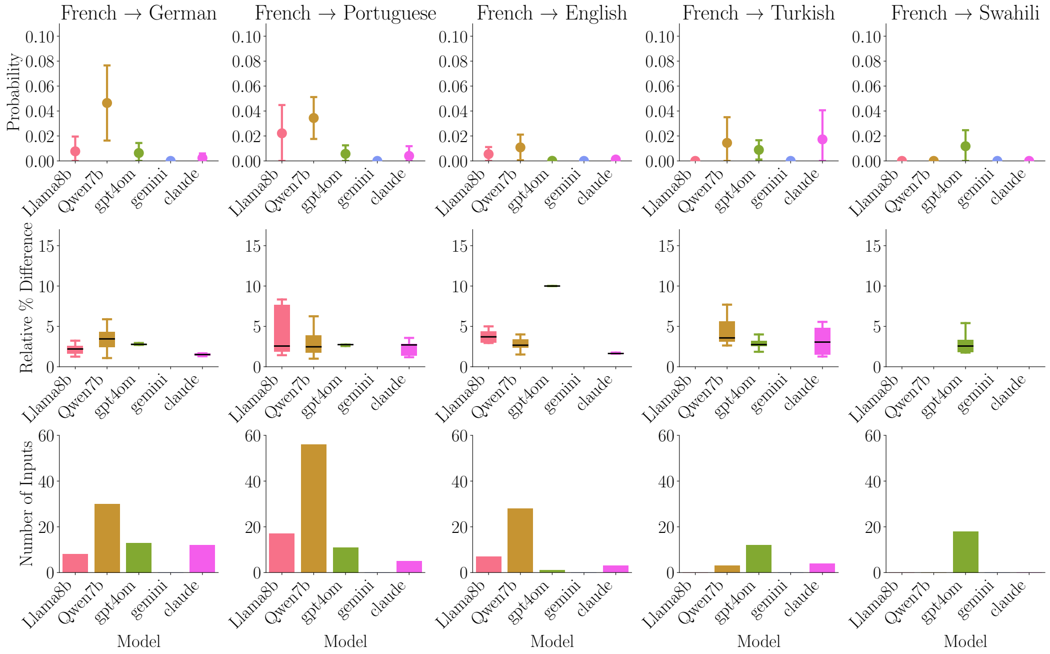


Figure 14: **Tokenization multiplicity in the translation task from French to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

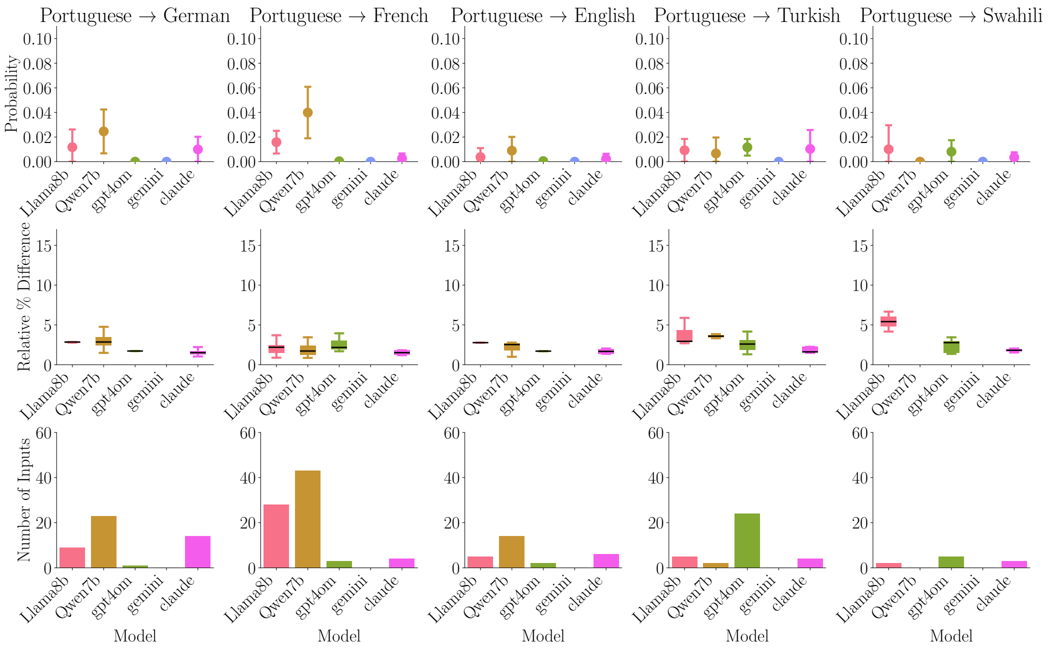


Figure 15: **Tokenization multiplicity in the translation task from Portuguese to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

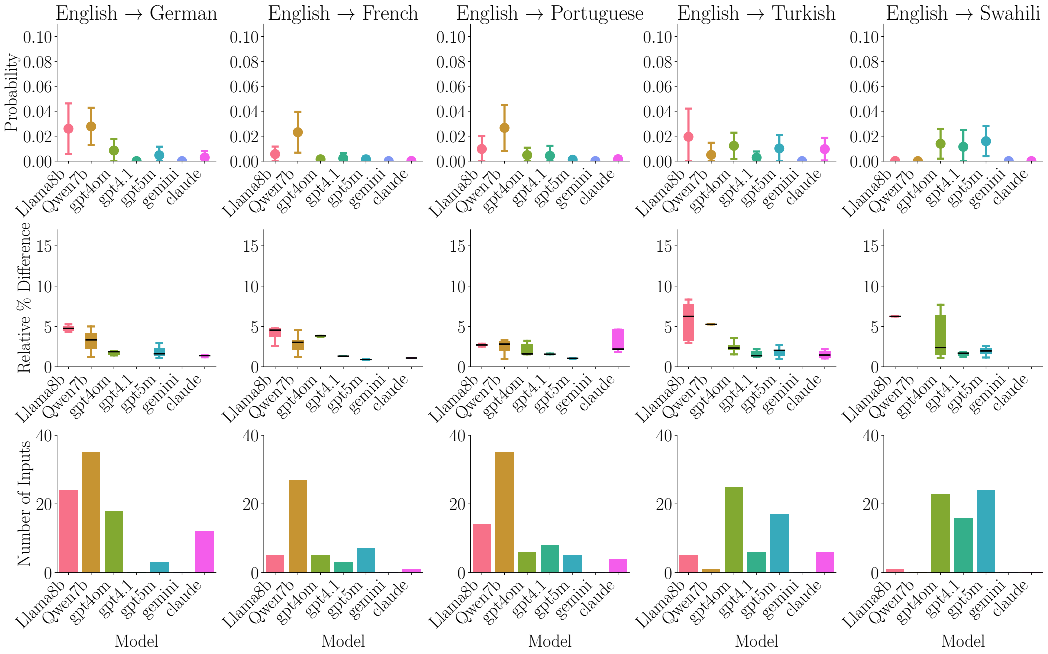


Figure 16: **Tokenization multiplicity in the translation task from English to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

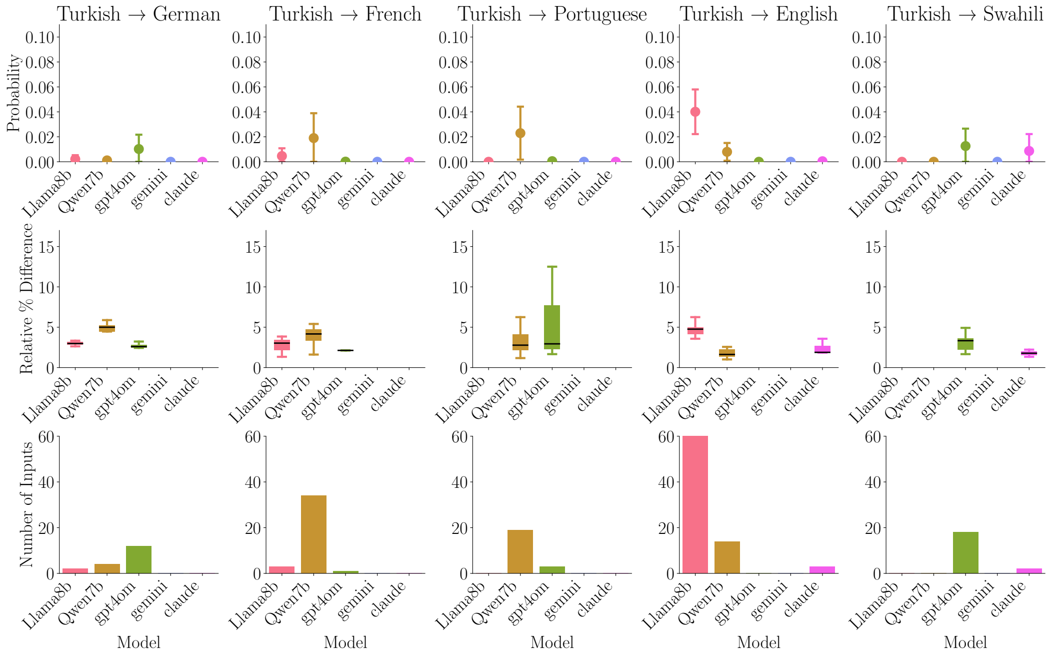


Figure 17: **Tokenization multiplicity in the translation task from Turkish to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

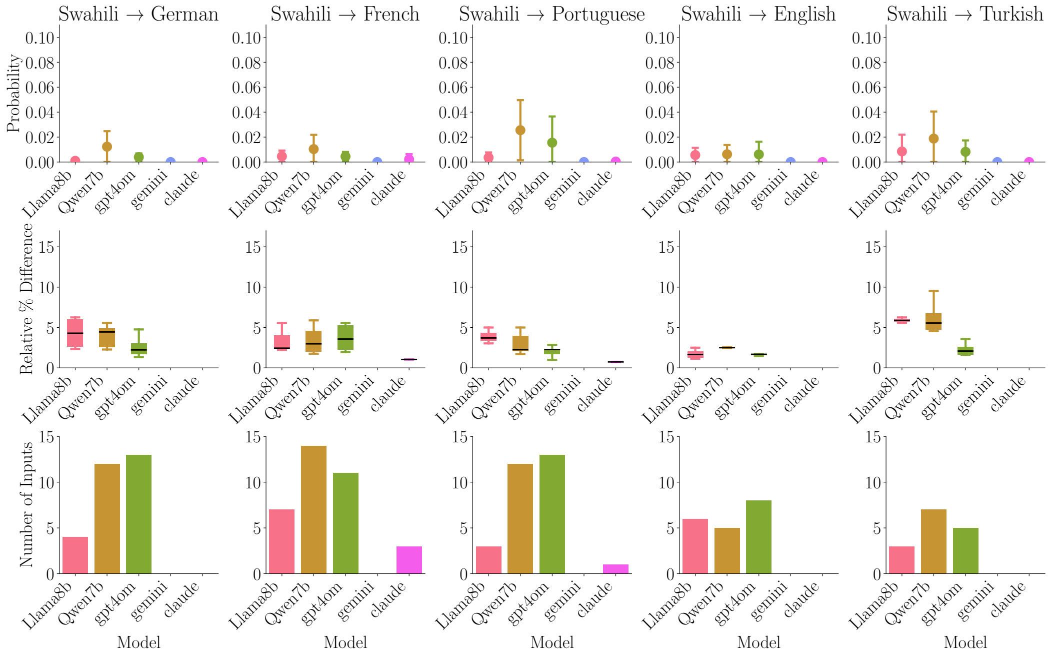


Figure 18: **Tokenization multiplicity in the translation task from Swahili to other languages.** The top row shows the probability of tokenization multiplicity, the middle row shows the magnitude of price variation, and the bottom row shows the number of inputs where we observed tokenization multiplicity. Each column corresponds to a different target language.

G.2 Additional Experimental Results on Canonical Generation

Language	Metric	Llama8B		Qwen7B	
		Standard	Canonical	Standard	Canonical
German	Quality score	0.72 ± 0.02	0.70 ± 0.02	0.73 ± 0.01	0.71 ± 0.01
	Time per token (s)	0.019	0.020	0.018	0.019
	Non-canonicity rate	6.1%	-	17.6%	-
French	Quality score	0.76 ± 0.02	0.74 ± 0.02	0.78 ± 0.02	0.76 ± 0.02
	Time per token (s)	0.020	0.020	0.018	0.018
	Non-canonicity rate	3.5%	-	8.6%	-
Portuguese	Quality score	0.72 ± 0.03	0.70 ± 0.03	0.78 ± 0.02	0.76 ± 0.02
	Time per token (s)	0.020	0.020	0.018	0.018
	Non-canonicity rate	3.9%	-	10.0%	-
Turkish	Quality score	0.57 ± 0.02	0.57 ± 0.02	0.61 ± 0.02	0.60 ± 0.01
	Time per token (s)	0.020	0.020	0.018	0.019
	Non-canonicity rate	27.6%	-	44.4%	-
Swahili	Quality score	0.60 ± 0.01	0.59 ± 0.01	0.43 ± 0.01	0.43 ± 0.01
	Time per token (s)	0.022	0.020	0.018	0.019
	Non-canonicity rate	18.6%	-	37.2%	-

Table 7: **Performance, (generation) time per token, and non-canonicity rate on the translation task.** The results comprise pairs of outputs generated with standard and canonical generation in all languages under the same source of randomness. For the time per token, confidence intervals were all smaller than 10^{-4}.

Language	Metric	Llama8B		Qwen7B	
		Standard	Canonical	Standard	Canonical
German	1 − edit distance	0.62 ± 0.04	0.61 ± 0.04	0.74 ± 0.02	0.72 ± 0.02
	Time per token (s)	0.019	0.023	0.018	0.018
	Non-canonicity rate	10.4%	-	19.0%	-
French	1 − edit distance	0.65 ± 0.04	0.64 ± 0.04	0.77 ± 0.02	0.76 ± 0.02
	Time per token (s)	0.019	0.022	0.018	0.018
	Non-canonicity rate	11.8%	-	18.4%	-
Portuguese	1 − edit distance	0.81 ± 0.03	0.72 ± 0.04	0.80 ± 0.02	0.76 ± 0.02
	Time per token (s)	0.020	0.022	0.018	0.019
	Non-canonicity rate	11.9%	-	16.8%	-
English	1 − edit distance	0.69 ± 0.06	0.68 ± 0.06	0.85 ± 0.02	0.83 ± 0.02
	Time per token (s)	0.019	0.020	0.018	0.019
	Non-canonicity rate	5.5%	-	8.6%	-
Turkish	1 − edit distance	0.63 ± 0.03	0.62 ± 0.03	0.70 ± 0.02	0.68 ± 0.02
	Time per token (s)	0.020	0.020	0.020	0.019
	Non-canonicity rate	26.8%	-	33.4%	-
Swahili	1 − edit distance	0.69 ± 0.03	0.68 ± 0.03	0.74 ± 0.02	0.74 ± 0.02
	Time per token (s)	0.020	0.020	0.021	0.019
	Non-canonicity rate	17.4%	-	19.4%	-

Table 8: **Performance, (generation) time per token, and non-canonicity rate on the spell checking task.** The results comprise pairs of outputs generated with standard and canonical generation in all languages under the same source of randomness. For the time per token, confidence intervals were all smaller than 10^{-4}.

Language	Metric	Llama8B		Qwen7B	
		Standard	Canonical	Standard	Canonical
German	Cosine similarity	0.84 ± 0.02	0.84 ± 0.02	0.89 ± 0.02	0.88 ± 0.02
	Time per token (s)	0.020	0.020	0.018	0.020
	Non-canonicity rate	5.7%	-	4.9%	-
French	Cosine similarity	0.90 ± 0.02	0.90 ± 0.02	0.93 ± 0.02	0.92 ± 0.02
	Time per token (s)	0.019	0.021	0.018	0.020
	Non-canonicity rate	2.2%	-	3.7%	-
Portuguese	Cosine similarity	0.92 ± 0.02	0.91 ± 0.02	0.96 ± 0.01	0.96 ± 0.01
	Time per token (s)	0.020	0.020	0.018	0.019
	Non-canonicity rate	2.2%	-	3.0%	-
English	Cosine similarity	0.93 ± 0.03	0.90 ± 0.04	0.96 ± 0.02	0.96 ± 0.02
	Time per token (s)	0.019	0.020	0.018	0.020
	Non-canonicity rate	0.2%	-	0.7%	-
Turkish	Cosine similarity	0.87 ± 0.01	0.88 ± 0.01	0.91 ± 0.01	0.90 ± 0.01
	Time per token (s)	0.020	0.020	0.019	0.019
	Non-canonicity rate	17.0%	-	17.0%	-
Swahili	Cosine similarity	0.85 ± 0.01	0.85 ± 0.01	0.85 ± 0.01	0.84 ± 0.01
	Time per token (s)	0.022	0.020	0.018	0.020
	Non-canonicity rate	10.3%	-	18.0%	-

Table 9: **Performance, (generation) time per token, and non-canonicity rate on the rephrasing task.** The results comprise pairs of outputs generated with standard and canonical generation in all languages under the same source of randomness. For the time per token, confidence intervals are not shown, as they were all smaller than 10^{-4}.

Language	Metric	Llama8B		Qwen7B	
		Standard	Canonical	Standard	Canonical
German	Accuracy	0.37 ± 0.06	0.37 ± 0.06	0.63 ± 0.05	0.62 ± 0.06
	Time per token (s)	0.020	0.020	0.018	0.020
	Non-canonicity rate	22.3%	-	29.4%	-
French	Accuracy	0.51 ± 0.06	0.50 ± 0.06	0.22 ± 0.15	0.22 ± 0.15
	Time per token (s)	0.020	0.020	0.019	0.019
	Non-canonicity rate	13.0%	-	0.6%	-
English	Accuracy	0.47 ± 0.09	0.47 ± 0.09	0.86 ± 0.12	0.86 ± 0.13
	Time per token (s)	0.020	0.020	0.019	0.019
	Non-canonicity rate	4.2%	-	0.7%	-
Spanish	Accuracy	0.57 ± 0.06	0.56 ± 0.06	0.58 ± 0.07	0.58 ± 0.07
	Time per token (s)	0.020	0.020	0.019	0.019
	Non-canonicity rate	14.1%	-	5.2%	-
Swahili	Accuracy	0.31 ± 0.05	0.32 ± 0.05	0.13 ± 0.03	0.13 ± 0.03
	Time per token (s)	0.021	0.020	0.018	0.019
	Non-canonicity rate	32.7%	-	42.5%	-

Table 10: **Performance, (generation) time per token, and non-canonicity rate on the MGSM task.** The results comprise pairs of outputs generated with standard and canonical generation in all languages under the same source of randomness. For the time per token, confidence intervals are not shown, as they were all smaller than 10^{-4}.