Train It and Forget It: Merge Lists are Unnecessary for BPE Inference in Language Models

Tomohiro Sawada [*], **Kartik Goyal**
Georgia Institute of Technology

Abstract

Standard Byte-Pair Encoding (BPE) tokenization compresses text by pairing a learned token vocabulary with a detailed merge list. Recent work has shown that this merge list exposes a potential attack surface for extracting information about language model's training data. In this paper, we explore the downstream impact of BPE inference algorithms that do not rely on this merge list at all, and hence differ from the encoding process during BPE training. To address this question, we investigate two broad classes of BPE inference schemes that differ from BPE application during training: a) targeted deviation from merge-lists including random merge orders, and various corruptions of merge list involving deletion/truncation, and b) non-targeted BPE inference algorithms that do not depend on the merge list but focus on compressing the text either greedily or exactly. Extensive experiments across diverse language modeling tasks like accuracy-based QA benchmarks, machine translation, and open-ended generation reveal that while targeted deviation from the merge lists exhibits significant degradation in language model performance, the non-targeted merge-list-free inference algorithms result in minimal impact on downstream performance that is often much smaller than expected. These findings pave way for simpler and potentially more privacy-preserving tokenization schemes that do not catastrophically compromise model performance.

1 Introduction

Byte-pair encoding (Gage, 1994; Sennrich et al., 2016; Kudo and Richardson, 2018; Radford et al.) is the standard algorithm used to tokenize input texts for large language models (LLMs). In practice, most BPE-based tokenizer implementations used for frontier language models[1] rely on a learned merge list to iteratively combine subword units into tokens during inference time. This BPE inference procedure is appealing because it mimics the merge application procedure during BPE training. However, dependence on the learned merge list exposes a vulnerability that might facilitate exploits to affect the model's downstream performance. Also, as shown in recent work (Hayase et al., 2024), these merge lists expose an attack surface where adversaries can steal information about the tokenizer's training data that is likely correlated with the LLM training data. Moreover, other works (Geiping et al., 2024) have shown that discrepancies between the tokenizer and LLM's training data can lead to "glitch tokens" which lead to generation failures thus, information about the tokenizer's training data can be used to finding and exploiting these glitches (Land and Bartolo, 2024). It is therefore undesirable to rely on the BPE merge list during the deployment of the associated language model.

Hence in this paper, we investigate the effectiveness of using alternative BPE inference algorithms that do not depend on the learned merge lists post hoc for large language models trained with merge-list dependent BPE tokenization. BPE vocabulary typically admits multiple possible segmentations of the input pretokens that can be obtained from a myriad of BPE inference schemes. However, as we show in our experiments, these schemes are not all equal and the standard merge-list dependent scheme is ideal because of its alignment with the BPE training procedure.[2] Specifically, we focus on two such algorithms that aim to optimally compress the input text: a) left-to-right encoding that greedily maximizes compression; and b) an exact maximal compression encoding algorithm to compress the

[*]Corresponding author: tsawada@gatech.edu.

[1]Most notably, the Hugging Face tokenizer codebase: https://github.com/huggingface/tokenizers

[2]Technically, the inference scheme used for tokenization of data during training of language models is the most ideal scheme. But in our experiments and general practice, the language models use the merge-list dependent BPE inference scheme.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 35045–35058
November 4-9, 2025 ©2025 Association for Computational Linguistics

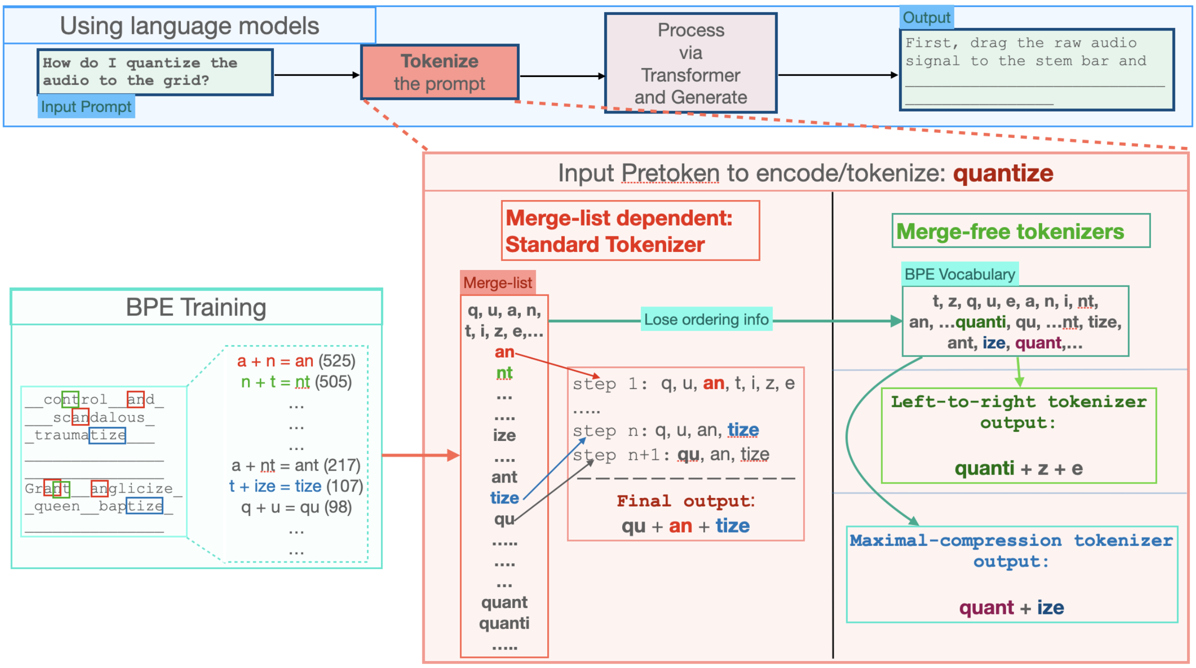

Figure 1: Illustration comparing merge-list-based and merge-list-free BPE algorithms elaborated in the pink expanded box. The pretoken "*quantize*" is tokenized by three different algorithms: a merge-list-based `standard` tokenizer (left) and two merge-list-free algorithms `left-to-right` (right-top) and `maximal-compression` (right-bottom). The ordered merge list is obtained from bigram statistics during BPE training. In contrast, merge-list-free algorithms only depend on the unordered BPE vocabulary, which contains less information about the training corpus.

input pretokens given the BPE vocabulary. We contrast the impact of these algorithms to a class of inference algorithms that arise by targeted manipulation of the vulnerable merge list which includes truncation/deletion of merges, random shuffling of ordered merges, and backing-off to single characters. On three diverse language modeling tasks – a) multiple-choice QA, b) conditional generation (machine translation), and c) open-ended generation – we observe that the targeted inference algorithms significantly degrade the downstream LLM performance, but the non-targeted algorithms focusing on compression do not negatively impact LLM performance, and even improve it in some cases. Finally, we conduct further quantitative and qualitative analysis to study this surprising pattern of results in greater detail.

Our contributions are: i) empirically support compression-focused inference algorithms for tokenization which ameliorate the security vulnerability arising from the dependence on merge-lists; ii) investigate the downstream effect of numerous BPE inference algorithms, including ones that exploit the merge-list vulnerability, that deviate from training on diverse language modeling tasks; and iii) shed light on the extent to which the non-deterministic encoding property of BPE documented in prior work is impactful in practice.

2 Training and Inference for BPE

Given a fixed BPE vocabulary, there are many possible encoding algorithms one can use to encode a pretoken. [3] Typically, BPE training produces merge-lists and inference also uses these merge lists in the same way as training to avoid mismatch and reduce ambiguity in segmentation. A merge-list-based BPE encoder is deterministic once the merge list is fixed; a vocabulary alone only specifies a family of possible deterministic encoders. Our focus in this paper is to explore *mismatched inference algorithms for BPE on a model pretrained with a merge-list-based inference scheme*. In this section, we review how standard BPE training and encoding process and describe the two alternate

[3] Given a BPE vocabulary, there are multiple ways to encode a given pretoken that are each deterministic functions of the input. Following the terminology in (Gastaldi et al., 2025), we call such tokenizers *non-deterministic*. See Section 5 for why this is a useful perspective.

merge-free BPE inference algorithms explored in this paper.

2.1 Training and Merge-list

BPE is a greedy compression algorithm that is trained on a corpus by repeatedly merging the most frequent pair of tokens in the training corpora, and recording the new merged token at each step into the BPE tokenizer vocabulary. In practice, each pretoken (space separated word) is processed individually across the corpus. This results in the vocabulary of the tokenizer.

A lesser known fact is that many standard BPE implementations also record the merge list, which is the ordered list of merges that were performed sequentially during the training process (see Figure 1). This list has strictly more information than the vocabulary alone because it contains the "training dynamics" of the tokenizer, namely a.) the splittings of the tokens (and hence the "dependencies" between them), and b.) the order of the merges. Recent work (Hayase et al., 2024) has shown that this information can be used to extract information about the tokenizer's training data, which is often correlated with the pretraining data of the language model. Thus, the tokenizer merge lists are potential attack surfaces which adversaries can exploit to extract information about the language model. In contrast, the BPE vocabulary does not include any information about the order of the merges, and is more difficult to use for attacks.

2.2 Merge-based BPE Encoding Algorithms

Standard implementations of BPE encodings use the merge list to encode pretokens returned by some pretokenization pipeline (which often returns a list of pretokens). To the best of our knowledge, there is no single consensus "standard BPE inference algorithm," even though popular libraries follow merge-list-based inference variants, e.g., Hugging Face Tokenizers[4], tiktoken[5], SentencePiece (BPE mode)[6], and fastBPE[7]. The tokenizer first attempts to match the pretoken with an element in the vocabulary. If there are no exact matches, the tokenizer then takes a list of merges from the merge list appearing in the pretoken, and subsequently applies the merges to the pretoken as illustrated in Figure 1. The primary motivation behind this scheme is to emulate the same compression process at inference time as in the training process so that the token distribution seen by the models at inference time is similar to the training distribution. In this paper, we call the algorithm described above the *merge-based* BPE encoding algorithm since it relies on the merge list at test time.

An important aspect of merge lists is their natural *hierarchical structure*. For example, if the bigram "an" is learnt at the first step of training, and the token "ant" is learnt at the seventh step by merging "an" and "t", then the token "ant" can only be used after applying the merge "a n", and so "ant" *is a child of* "an". This is a key property of merge lists. We revisit this in our merge-list perturbation based experiments – when we delete a symbol from the merge list, we must also delete all its children since they are no longer reachable during the standard BPE encoding process. As noted above, the merge lists provide a security risk which can have severe consequences to model providers. Our work shows that it is possible to encode text by patching this vulnerability *while maintaining downstream performance*. Moreover, our method does *not* require retraining the language model on the new tokenizer, and can be applied post-hoc to any existing language model.

2.3 Non-targeted merge-list-free BPE inference algorithms

Given a BPE vocabulary, we can encode a pretoken without relying on the merge list. We explore two merge-list-free algorithms that focus on compression and integrate cleanly into the LM inference pipeline. These likely behave well because BPE training can be interpreted (Zouhar et al., 2023) as implicitly prioritizing compression in a greedy manner. We call them *non-targeted* be-

[4] `https://github.com/huggingface/tokenizers/blob/ee2c5708bdce9d6610fa74faeb22cf6297c6390a/tokenizers/src/models/bpe/model.rs#L382C5-L468C1`

[5] `https://github.com/openai/tiktoken/blob/4560a8896f5fb1d35c6f8fd6eee0399f9a1a27ca/src/lib.rs#L17-L83`

[6] `https://github.com/google/sentencepiece/blob/273449044caa593c2fd7eb7550cb3ab2cff93f1a/src/bpe_model.cc#L38-L202`

[7] `https://github.com/glample/fastBPE/blob/036711f8fdc3265d64e8e123a0761be12c5a8e74/fastBPE/fastBPE.hpp#L581-L630`

cause they do not manipulate the learned merge list. In this paper, we call a merge-list-free encoding algorithm *performant* if it achieves comparable or better downstream performance as the standard merge-based encoding algorithm.

2.3.1 Left-to-right greedy encoding

The left-to-right encoding algorithm is a simple and efficient procedure for encoding a pretoken. Given a pretoken, we look for the longest prefix of the pretoken that is in the vocabulary, and we output that prefix as a token. We then repeat this process for the remaining suffix of the pretoken. For example, given the pretoken "quantize" and the vocabulary provided in 1, the left-to-right encoding algorithm chooses the token "quanti" (as opposed to "quant" or "qu") since it is the longest prefix in the vocabulary. The suffix "ze" is then encoded as "z" and "e" since the string "ze" is not in the vocabulary.

This is a natural candidate for a performant merge-list-free encoder. Since BPE learns tokens greedily during training, it is plausible that left-to-right encoding achieves a similar level of compression.

2.3.2 Maximal compression encoding

Prior work (Goldman et al., 2024a) has shown that compression during LLM *pretraining* correlates strongly with downstream performance. It is therefore natural to ask whether better compression at inference time leads to better downstream performance. To address this question, we consider the *maximal compression encoding algorithm*. Given a pretoken, we look for the combination of tokens in the vocabulary which gives the highest compression of the pretoken.

For example, if we have the pretoken "quantize" and the vocabulary provided in 1, the string "quantize" is not an element in the vocabulary, so the shortest encoding must contain at least two tokens. From manual inspection, we see that "quant" and "ize" are both in the vocabulary, so the maximal compression encoding algorithm chooses this split.

A naive implementation has exponential time complexity in the pretoken length, but dynamic programming reduces this to quadratic time [8]. In practice, pretokens are short after pretokenization.

[8] See Appendix, Algorithm 1.

2.4 Other merge-list-free encoding algorithms

Although there are many other merge-free inference algorithms, many of them do not compress the prompt as well as the ones discussed above. The most trivial one is the character-based encoding algorithm: this breaks the pretoken into characters and outputs them as tokens. This encoding method has the *worst* compression for a given piece of prompt, and is thus the opposite of the maximal compression encoding algorithm.

As described in the subsequent sections, we observe that the compression-oriented inference algorithms, especially the left-to-right greedy encoding algorithm, have comparable downstream performance to the standard merge-based encoding algorithm, while the character-based encoding algorithm, although also merge-free, performs significantly worse.

3 Impact of Training-Inference Mismatch on LM Performance

In this section, we describe our empirical findings on the impact of different tokenization schemes on downstream LM performance. We not only compare the merge-free non-targeted compression-based inference algorithms to the standard tokenization algorithm described above, but we also investigate other tokenization schemes that explicitly seek to exploit and manipulate the vulnerabilities offered by a publicly available merge-list. We perform extensive investigation on three diverse LM-based tasks as described below. The central question we aim to explore is the nature of the impact of the mismatch between training and inference time tokenization procedures.

3.1 Experimental Setup

We evaluate an LLM on three diverse kinds of tasks: multiple-choice QA tasks that require very short form generation after encoding the question prompt, a longer conditional generation task of machine translation that involves processing a prompt with the source text and generating target text, and a fully open-ended generation task that focuses on completion based on context to be encoded. We process the prompts with the different encoding schemes, but generate with the full vocabulary. It must be noted that the choice of tokenization inference does not affect generation with a BPE-based tokenizer.

We choose to focus on the

`Qwen-2-7B-Instruct` model (Yang et al., 2024) for our experiments. The choice of model is motivated by the need for a model with a sizable vocabulary size to experiment with different ranges of corruptions) and a tokenizer which was trained using the HF tokenizer (as opposed to tiktoken). [9] The Qwen-2 tokenizer has 151645 tokens in its vocabulary, of which 255 are single character tokens. Unless noted otherwise, the Qwen-2 tokenizer will be referred to as the "standard" tokenizer (as opposed to the "custom" tokenizers obtained by either using a different encoding algorithm or by corrupting the merge list).

3.1.1 MCQA tasks

For the accuracy-based tasks, we evaluate the model on two popular Q&A benchmarks: MMLU (Hendrycks et al., 2021) and ARC-Easy/Challenge (Clark et al., 2018).

3.1.2 Conditional Generation: Machine Translation

We consider the effect of different tokenizations on the semantic correctness of the generated text by testing it on the task of machine translation. We evaluate the performance of the model on WMT16 Czech→English and WMT15 German→English test sets, and report COMET (Rei et al., 2020) as our primary metric in the main paper. We relegate BLEU (Papineni et al., 2002) and METEOR (Banerjee and Lavie, 2005), along with xCOMET and CometKiwi, to the Appendix for completeness. BLEU is computed on detokenized outputs.

3.1.3 Open-ended Generation

We evaluated the open-ended generation capabilities of the model by prompting it with abstracts from scientific papers. The prompts were generated by extracting the first five sentences from the full text in the "Semantic Scholar Open Research Corpus" (Lo et al., 2020), a text corpus consisting of research papers extracted from Semantic Scholar. To ensure the quality and diversity of the prompts, we took 5,899 examples from the corpus across multiple academic fields. The text corpora was chosen due to the high concentration of domain-specific pretokens which are likely to be sensitive to tokenization.

We then measured how much the generated text deviated from the original human-written distribution by measuring the MAUVE score (Pillutla et al., 2023) between the two sets of texts.

3.2 Targeted Tokenization

As mentioned above, we measure the impact on the downstream LM performance when the inference algorithms target to manipulate merge-list obtained via tokenizer training. We deliberately corrupt the merge list of the tokenizer and measure the performance degradation. To understand the sensitivity of LLM inference on the merge list, we ran inference using tokenizations generated from a corrupted merge list. The merge list gives a fine-grained interface for controlling the encoding of the model (as opposed to the choice of encoding algorithms which are *qualitatively* different from one another). These experiments can also help us understand to what extent the manipulation of the merge list (by for example, a malicious insider) can be used to sabotage the generation capability of the model. We corrupt the tokenizer in the following ways:

Truncation: Since the merge lists are generated in the order in which the merges are learned, we consider the effect of removing the less common merges (learned last during training) by deleting the last N merges from the merge list.

Deletion: We also consider the effect of *random deletion* of merges since the merges important for downstream performance may not be concentrated in a particular region within the merge list. For random deletions, we first choose an initial set of deletions (the "initial set") and delete all merges which depend on these seeds (the "number of deletions"). To generate our random deletion tokenizers, we've fixed a random seed, chose an increasing number of initial deletions, and measured the performance of the model for each of these settings. (This is why the number of deletions is not a clean number for all of our random deletion experiments.)

Merge Shuffle: We also consider a merge-based tokenization where at runtime, we randomly shuffle the merge list being applied to the pretoken. For example, the standard encoding algorithm 1 may tokenize the pretoken "quantize" by successively applying the merges "a n", "z e", "i ze", "t ize", and "q u", in this order, resulting in the tokenization "qu

[9] These desiderata eliminates other popular models such as OLMo-7B and Llama-3. The former uses the GPT-NeoX tokenizer, which has 50k tokens in its vocabulary, and the latter uses the tiktoken tokenizer.

The tiktoken tokenizer is a proprietary tokenizer developed by OpenAI for their models. Their merge lists do not strictly adhere to the requirements we described in the previous section.

an tize". The random shuffle encoding algorithm may instead apply the merges "u a", "n t", "q ua", "nt i", and "ze" (assuming all of these appear in the merge list somewhere), resulting in the tokenization "qua nti ze". Throughout our experiments, we have a fixed random seed which determines how the merge list is shuffled.

The random shuffle encoding results in a drastically different token distribution at inference time compared to the standard encoding algorithm. This provides a natural baseline where we expect the generation capability of the model to be significantly degraded.

Character Level: As described above, we also consider the baseline of splitting pretokens into individual characters.

	Accuracy-based Tasks		Machine Translation		OEG.
Tokenizer	ARC	MMLU	De→En	Cz→En	MAUVE
Standard	0.869	0.656	0.502	0.685	0.904
Merge shuffle	0.853	0.617	0.478	0.633	0.245
Character-level	0.860	0.624	0.479	0.520	0.399
Random deletion	0.860	0.628	0.483	0.531	0.170

Table 1: Evaluation results for merge-list-based corruption tokenizers on the accuracy-based tasks (ARC and MMLU) and machine translation (COMET for WMT De→En and Cz→En). The corrupted tokenizers do not suffer as much for accuracy-based tasks compared to longer generation tasks. The random deletion tokenizer was obtained by randomly deleting 149 802 merges from the standard tokenizer's merge list. "OEG." stands for "Open-ended Generation."

Observing the results in Table 1, we see that the corruption doesn't seem to affect the MCQA tasks much but it shows significant degradation in MT and open-ended generation under corruption. Although the prompts in accuracy-based benchmarks are long enough to have different tokenizations under our scheme, the generation length is not long enough to show substantial differences in performance. The merge shuffle corruption consistently performs at least as bad as, if not worse than, the character-level corruption. This suggests that severe corruption to the merge lists can essentially do away any benefits of subword tokenization, and the model may as well use a character-level tokenization.

In Figure 2, we investigate the relationship between the effect on downstream performance and severity of corruption. We observe that both semantic and n-gram metrics are not too sensitive to mild corruption on a per-example level. As the corruption levels cross a threshold (merge shuffle, char-level, aggressive deletion), the drop in performance is noticeably significant. In fact, we've observed that performance is quite stable even for "medium-sized" deletions (107060 and 115604). This seems to suggest that the model's performance relies primarily on "highly-trained" tokens which are only destroyed for very aggressive corruptions. It could also be the case that large portions of BPE vocabulary are never used for practical purposes indicating the existence of many undertrained tokens in the vocabulary. It is also interesting to note that the decline for the random deletion tokenizer is more steady in the machine translation task compared to the accuracy-based tasks. This robustness is likely due to the fact that the model is generating longer text in the machine translation task.

Overall on manual inspection, the degradation of generated output exhibits unnatural syntactic choices (e.g., characters separated out by spaces) which causes drops in BLEU and MAUVE.

3.3 Non-targeted Tokenization

As described above, we compare compression-based merge-free algorithms against the standard algorithm. These algorithms either greedily or exactly maximize compression of the pretoken given the BPE vocabulary.

	Accuracy-based Tasks		Machine Translation		OEG.
Tokenizer	ARC	MMLU	De→En	Cz→En	MAUVE
Standard	0.869	0.656	0.502	0.685	0.904
Maximal Compression	0.863	0.678	0.494	0.633	0.927
Left to right	0.903	0.705	0.495	0.633	0.985

Table 2: Evaluation results for merge-list-free tokenizers on the accuracy-based tasks (ARC and MMLU), machine translation (COMET for WMT De→En and Cz→En), and the open-ended generation task. The left-to-right tokenizer maintains performance or even outperforms the standard tokenizer on QA and OEG; small degradations are observed on MT (see text). The maximal compression also largely maintains the standard tokenizer's performance. "OEG." stands for "Open-ended Generation."

In Table 2, we observe that left-to-right and maximal compression tokenization schemes maintain performance on accuracy-based QA and OEG, with left-to-right even improving ARC/MMLU and MAUVE. For machine translation, however, COMET shows modest but consistent drops relative to the standard tokenizer: for De→En, 0.5017

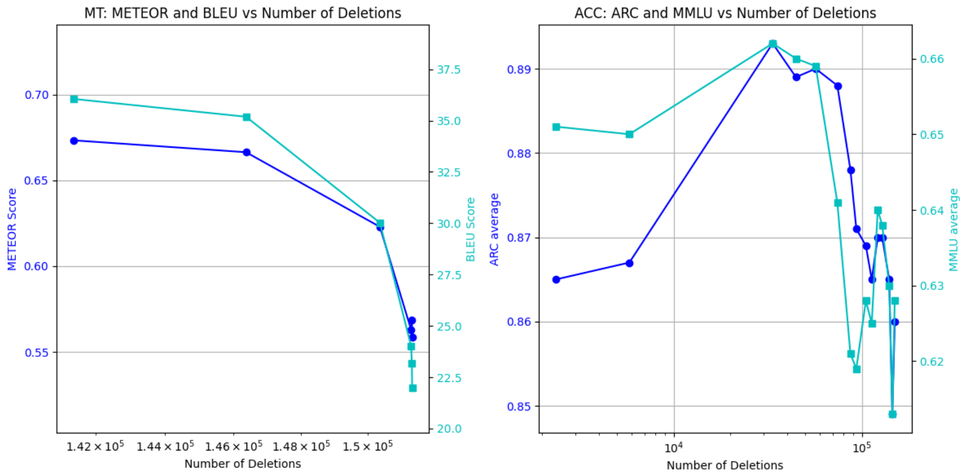

Figure 2: Performance of different random deletion tokenizers on the accuracy-based tasks (ARC and MMLU) and the machine translation task. For both datasets, the performance degrades after around 70k deletions.

(standard) vs. 0.4953 (left-to-right) and 0.4944 (maximal); for Cz→En, 0.6853 (standard) vs. 0.6325/0.6328. These differences are potentially meaningful, particularly for Cz→En. In Appendix Table 6 and Table 5, BLEU and METEOR broadly reflect the same directionality, though COMET appears more sensitive on Cz→En. This aligns with COMET's semantic focus, while BLEU/METEOR capture n-gram surface deviations. Overall, these results indicate that merge-list-free, compression-based encoders are robust across tasks but can induce small MT degradations, especially in morphologically richer settings.

This is because when we measure how differently the prompts are encoded under various tokenization schemes compared to the standard tokenizer, we find that the merge-free tokenizers differ in the encoding of every single prompt in the open-ended generation task. Table 3 shows the average edit distances between the merge-based standard tokenizer encodings and encodings from the other tokenizers. We observe that the left-to-right and maximal compression encodings are less distant than other corruption-based tokenizers. Though, we also notice that they have higher perplexity on the prompts than the standard merge-based tokenizer. This indicates that the compression-based approaches use potentially unconventional and undertrained tokens, but these effects are overcome by the model's robustness to specific kinds of typos and over-segmentations associated with compression-based algorithms.

4 Characterizing Bad Tokenization

We analyze why different tokenizations lead to different downstream performance, complementing the aggregate results in Section 3. We present quantitative indicators of sensitivity/robustness and qualitative patterns in tokenization differences.

4.1 Quantitative indicators of sensitivity and robustness

Why do some tokenizations lead to better performance than others? One hypothesis is the presence of *undertrained tokens*, i.e., vocabulary elements that occurred infrequently during pretraining. Past work (Land and Bartolo, 2024) shows such tokens can cause undesirable behavior, e.g., failures to follow instructions. Following (Land and Bartolo, 2024), we use the minimum token-embedding norm as a proxy for undertrainedness.[10]

Prior work (Bigelow et al., 2024) also shows that changing a single token can drastically alter generations, suggesting even a small number of undertrained tokens in a prompt may lead to poor behavior. To study this, we examine "surprising" cases where large tokenization deviations yield small performance changes (robust) and where small tokenization deviations yield large performance changes (sensitive). For each generated sequence L, we compute

$$\min_{t \in L} \|E(t)\|, \tag{1}$$

where $\|\cdot\|$ is the norm and $E(t)$ is the token

[10] If a token appears rarely during pretraining, its embedding sees fewer updates and tends to remain closer to the origin.

Tokenizer	Jaccard	Levenshtein	Edit	Perplexity
Standard	0.000	0.000	0.000	83.798
Left to right	0.226	29.645	0.165	95.891
Maximal Comp.	0.196	24.740	0.139	155.751
Merge Shuffle	0.918	692.000	0.959	131.400
Character-level	0.925	796.987	0.964	58.212
Random Deletion	0.927	800.719	0.966	92.734
Truncation	0.889	455.775	0.884	97.202

Table 3: Perplexity scores and prompt metrics (Jaccard similarity, Levenshtein distance, edit distance) between different tokenization approaches and standard tokenization.

	Cluster Size		Min Norm	
	Robust	Sensitive	Robust	Sensitive
Random deletion	420	157	2891.22	1609.89
Character-level	430	115	2891.01	1606.73
Merge shuffle	415	128	2837.74	1539.82
Left to right	406	64	2748.35	1626.64
Maximal compression	397	60	2720.20	1597.86

Table 4: Cluster size and minimum token embedding norm computed over each cluster. We analyze the WMT15 de-en test set (2998 samples). Notice that a) the robust cluster tends to be larger for non-targeted tokenizers compared to targeted tokenizers, and b) the norm for sensitive cluster is consistently smaller than robust cluster.

embedding. In Table 4, sensitive clusters are consistently larger for targeted tokenizations than for merge-list-free ones, indicating targeted schemes more often induce small perturbations that cause large performance deviations. Sensitive clusters also exhibit smaller minimum norms than robust clusters, consistent with a higher incidence of undertrained tokens.

4.2 Qualitative patterns in tokenization differences

We further examine token-level edit distance (Levenshtein distance between token-ID sequences) relative to the standard tokenizer for open-ended generation. Figure 3 summarizes the distributions; merge-list-free (non-targeted) algorithms produce smaller encoding differences than targeted schemes.

Qualitatively, low-distance prompts cover diverse scientific domains and use simpler language; high-distance prompts are often biomedical with hyperspecific jargon and rare terms, consistent with more rarely trained tokens. This aligns with the quantitative evidence that sensitive cases are associated with smaller embedding norms.

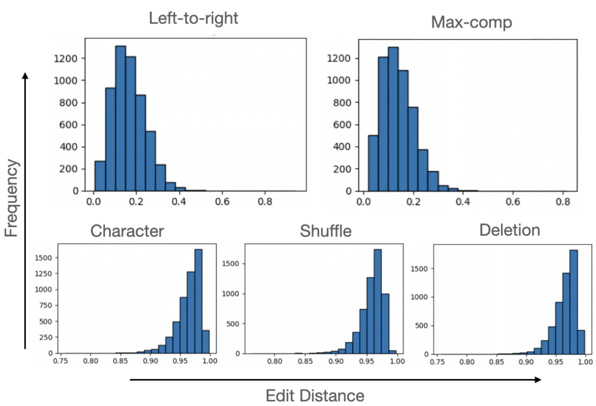

Figure 3: Distribution of token-level edit distance between standard and alternative tokenizers for open-ended generation (S2ORC subset; N noted in text). Top: non-targeted merge-list-free schemes. Bottom: targeted schemes. Lower is closer to standard.

5 Related Work

While we focus on BPE inference algorithms that ameliorate security vulnerabilities associated with merge-lists (Hayase et al., 2024), the non-deterministic property [11] of tokenization algorithms (Kudo and Richardson, 2018; Sennrich et al., 2016; Mielke et al., 2021) in general—identified in several prior works (Cao and Rimell, 2021; Gastaldi et al., 2025)—forms the crux of our motivation. The symbols in the vocabulary can give rise to multiple possible segmentations for a given word/pretoken, some of which may be *non-canonical*. This is a useful perspective because it opens the possibility for more sophisticated tokenization/inference algorithms with desirable properties. Several recent works have explored this from the perspective of treating tokenization of a character string as a latent variable and marginalizing over them to compute the "true" distribution over character strings. For instance, (Geh et al., 2025) proves that marginalizing over all tokenizations to find the "true" string probability is computationally hard. Critically, they also provide the surprising empirical result that even though non-canonical tokenizations have negligible probability mass, aggregating them improves performance on downstream QA tasks. (Vieira et al., 2025a) provides an efficient beam-search algorithm to approximate this marginalization, enabling the conversion of token-level models into more principled character-level ones. In contrast, (Vieira et al., 2025b) argues that non-canonical tokenizations are a probabilistic flaw. They propose inference methods and architectural changes to force the model to only generate canonical sequences, thereby correcting the probability distribution. Our paper differs from the concurrent work in that we explored the empirical effect of non-canonical tokenization for the prompt inputs for downstream tasks (which are not *a priori* obvious from the marginalization perspective above). Our work also explicitly identifies tokenizations which performed well in downstream tasks. This raises the question: what mechanisms are at play behind transfer of training on canonical tokenization to other alternative schemes? We hope to explore this in future work.

While much work has studied the effect of training different types of tokenizers/segmenters *and* models based on those tokenizers (Goldman et al., 2024b; Saleva and Lignos, 2023), we instead focus on evaluating different BPE inference schemes on pretrained tokenizers and models with the standard BPE approach. While training models (Provilkov et al., 2020) with different tokenization schemes in general does not affect downstream performance significantly, in our setting of training–inference mismatch we observe significant performance degradation with certain algorithms.

Related to our work, (Uzan et al., 2024) also study different BPE inference algorithms, but they limit their analysis to intrinsic tokenization metrics like cognitive plausibility (Beinborn and Pinter, 2023) and morphology (Bostrom and Durrett, 2020) and do not investigate their downstream impact on model performance. Our finding that algorithms like left-to-right and maximal compression do not result in significant performance degradation despite encoding the prompts differently is also related to recent findings that LLMs have an implicit lexicon of pretokens (Kaplan et al., 2025) and are robust to typos (Cao et al., 2023).

6 Conclusion

In light of security vulnerabilities associated with inference-time usage of the merge-list learned during BPE training, we explored alternative merge-free algorithms for BPE inference on pretrained models. We found that although arbitrary and targeted inference-time deviations from standard BPE hurt downstream LM performance significantly, surprisingly the non-targeted compression-based merge-free algorithms maintained or even improved it. This suggests potential overlap in the implicit objectives of BPE training and these merge-free algorithms paving way for more secure tokenization schemes for language models.

7 Limitations

The primary limitation of our work is that while we have articulated the need for merge-list-free BPE inference algorithms and have provided empirical evidence for two such inference algorithms focusing on compression across a diverse set of LM tasks, it is not clear that the algorithms investigated are the optimal algorithms for merge-free inference that preserves performance across *all* domains and languages. Relatedly, we only have empirical support from our experiments and prior works for concluding that left-to-right and maximal compression algorithms preserve performance—possibly because the original BPE training procedure implicitly greedily optimizes (Zouhar et al., 2023) for compression and breaks ties in a left-to-right manner for most languages. We do not have theo-

[11] See Section 2 for the use of the term "non-deterministic tokenization."

retical support and guarantees for this conjecture, and our findings might not hold for small amounts of data in low-resource languages, especially with a non-monotonic or a non-left-to-right writing order. Finally, while our recommendation might eliminate data inference and other security vulnerabilities directly related to merge-lists, they still would not defend against other kinds of attacks based on tokenization such as those focusing on finding and exploiting *glitch* tokens.

8 Ethical Considerations

While we recommend defending against vulnerabilities associated with merge lists during deployment by not using them, this would also result in less transparency. It can be argued that publicly available merge-lists possible allow data-mixture inference and it might be desirable in certain cases because of transparency and auditability reasons. However, depending on the context, it can also be argued that LMs should be protected from the security vulnerabilities posed by publicly available merge-lists. We recognize that our recommendation applies for the latter contexts and doesn't apply in contexts that disproportionately prioritize transparency.

References

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Lisa Beinborn and Yuval Pinter. 2023. Analyzing cognitive plausibility of subword tokenization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4478–4486, Singapore. Association for Computational Linguistics.

Eric Bigelow, Ari Holtzman, Hidenori Tanaka, and Tomer Ullman. 2024. Forking paths in neural text generation. *Preprint*, arXiv:2412.07961.

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

Kris Cao and Laura Rimell. 2021. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Qi Cao, Takeshi Kojima, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Unnatural error correction: GPT-4 can almost perfectly handle unnatural scrambled text. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8898–8913, Singapore. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint*. ArXiv:1803.05457 [cs] version: 1.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal archive*.

Juan Luis Gastaldi, John Terilla, Luca Malagutti, Brian DuSell, Tim Vieira, and Ryan Cotterell. 2025. The foundations of tokenization: Statistical and computational concerns. In *The Thirteenth International Conference on Learning Representations*.

Renato Lui Geh, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van den Broeck. 2025. Where is the signal in tokenization space? *Preprint*, arXiv:2408.08541.

Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. 2024. Coercing llms to do and reveal (almost) anything. *Preprint*, arXiv:2402.14020.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024a. Unpacking Tokenization: Evaluating Text Compression and its Correlation with Model Performance. *arXiv preprint*. ArXiv:2403.06265 [cs].

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. 2024b. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2274–2286, Bangkok, Thailand. Association for Computational Linguistics.

Jonathan Hayase, Alisa Liu, Yejin Choi, Sewoong Oh, and Noah A Smith. 2024. Data Mixture Inference: What do BPE Tokenizers Reveal about their Training Data?

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. *arXiv preprint*. ArXiv:2009.03300 [cs].

Guy Kaplan, Matanel Oren, Yuval Reif, and Roy Schwartz. 2025. From tokens to words: On the inner lexicon of LLMs. In *The Thirteenth International Conference on Learning Representations*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Sander Land and Max Bartolo. 2024. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. *Preprint*, arXiv:2405.05417.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. 2020. S2ORC: The Semantic Scholar Open Research Corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online. Association for Computational Linguistics.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. *arXiv preprint*. 2.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Krishna Pillutla, Lang Liu, John Thickstun, Sean Welleck, Swabha Swayamdipta, Rowan Zellers, Sewoong Oh, Yejin Choi, and Zaid Harchaoui. 2023. MAUVE Scores for Generative Models: Theory and Practice. *arXiv preprint*. ArXiv:2212.14578 [cs].

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners.

Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020. COMET: A Neural Framework for MT Evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.

Jonne Saleva and Constantine Lignos. 2023. What changes when you randomly choose BPE merge operations? not much. In *Proceedings of the Fourth Workshop on Insights from Negative Results in NLP*, pages 59–66, Dubrovnik, Croatia. Association for Computational Linguistics.

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

Omri Uzan, Craig W. Schmidt, Chris Tanner, and Yuval Pinter. 2024. Greed is All You Need: An Evaluation of Tokenizer Inference Methods. *arXiv preprint*. ArXiv:2403.01289 [cs].

Tim Vieira, Ben LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell. 2025a. From language models over tokens to language models over characters. *Preprint*, arXiv:2412.03719.

Tim Vieira, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian DuSell, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J. O'Donnell, and Ryan Cotterell. 2025b. Language models over canonical byte-pair encodings. *Preprint*, arXiv:2506.07956.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

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

A Appendix

Algorithm 1 Dynamic programming for maximal-compression BPE encoding. Given an input string s and a BPE vocabulary represented as a prefix trie rooted at `root`, this procedure finds the shortest sequence of token IDs whose concatenation exactly matches s. We maintain a one-dimensional array `dp[0..n]` where dp[i] holds the best encoding (minimal number of tokens) for the prefix $s[0..i-1]$. At each position i, we traverse the trie from the root to extend all valid tokens starting at i, updating dp[j+1] whenever we discover a shorter encoding ending at j. Time complexity $\mathcal{O}(n^2)$, space complexity $\mathcal{O}(n)$.

```
procedure MaxCompBPEEncode(s, root)
    n ← |s|
    dp ← [None]_{0..n}
    dp[0] ← []
    for i ← 0 to n − 1 do
        if dp[i] ≠ None then
            node ← root
            for j ← i to n − 1 do
                if s[j] ∉ node.children then
                    break
                node ← node.children[s[j]]
                if node.token_id is defined then
                    candidate ← dp[i] || node.token_id
                    if (dp[j + 1] = None) ∨ |candidate| < |dp[j + 1]| then
                        dp[j + 1] ← candidate
    return dp[n]
```

Tokenizer	COMET	xCOMET	CometKiwi
Standard	0.501667554	0.5854197109	0.3723556075
Maximal Compression	0.4944257367	0.5497664318	0.3476544334
Left to right	0.4953334777	0.5668370956	0.3942931857
Merge Shuffle	0.4775965853	0.4836160445	0.2200489447
Character-level	0.479254821	0.5089406749	0.2534056265
Random Deletion	0.482456316	0.5226766439	0.2672921025

Table 5: German-to-English machine translation results (COMET primary; xCOMET, CometKiwi in Appendix for completeness).

Tokenizer	COMET	xCOMET	CometKiwi	BLEU	METEOR
Standard	0.6852737008	0.5452000509	0.3664651487	18.3668	0.4682134426
Maximal Compression	0.6327833553	0.454005471	0.2956552479	12.9995	0.4124162695
Left to right	0.632486759	0.4565238956	0.297760896	12.8238	0.4125467474
Merge Shuffle	0.632486759	0.2627696861	0.1404431969	3.4871	0.2103303214
Character-level	0.5194693237	0.2788883335	0.1708871849	5.2251	0.2400589537
Random deletion	0.5312855418	0.288390215	0.1817652996	5.317	0.2511355355

Table 6: Czech-to-English machine translation results. BLEU is computed on detokenized outputs.

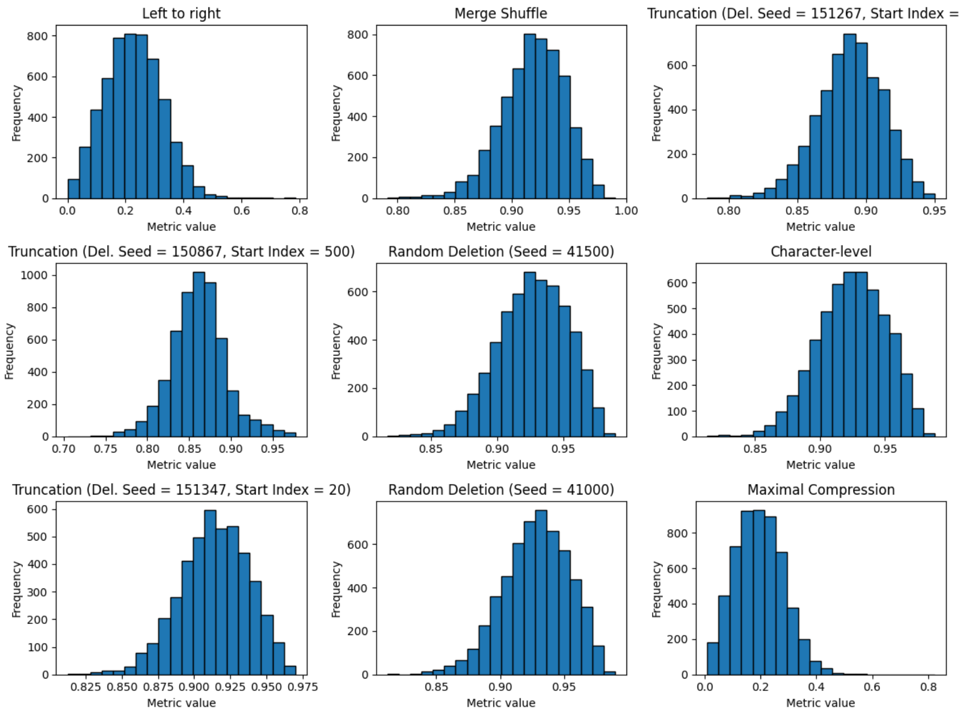

Figure 4: Jaccard distance between the tokenization of the Semantic Scholar prompts obtained from the standard tokenizer and custom tokenizers.

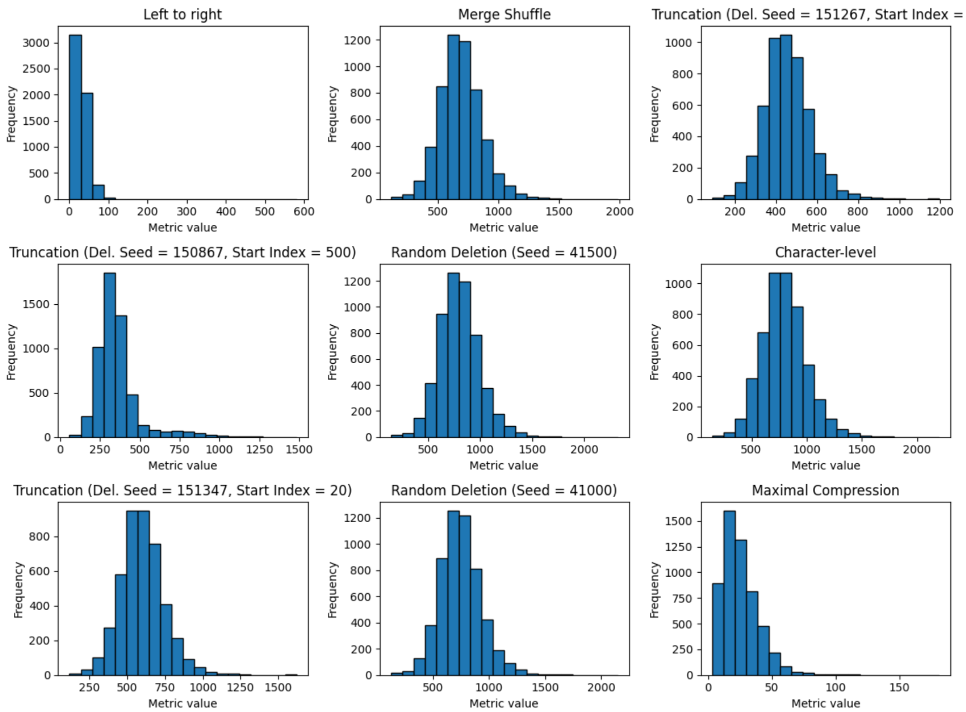

Figure 5: Levenshtein distance between the tokenization of the Semantic Scholar prompts obtained from the standard tokenizer and custom tokenizers.

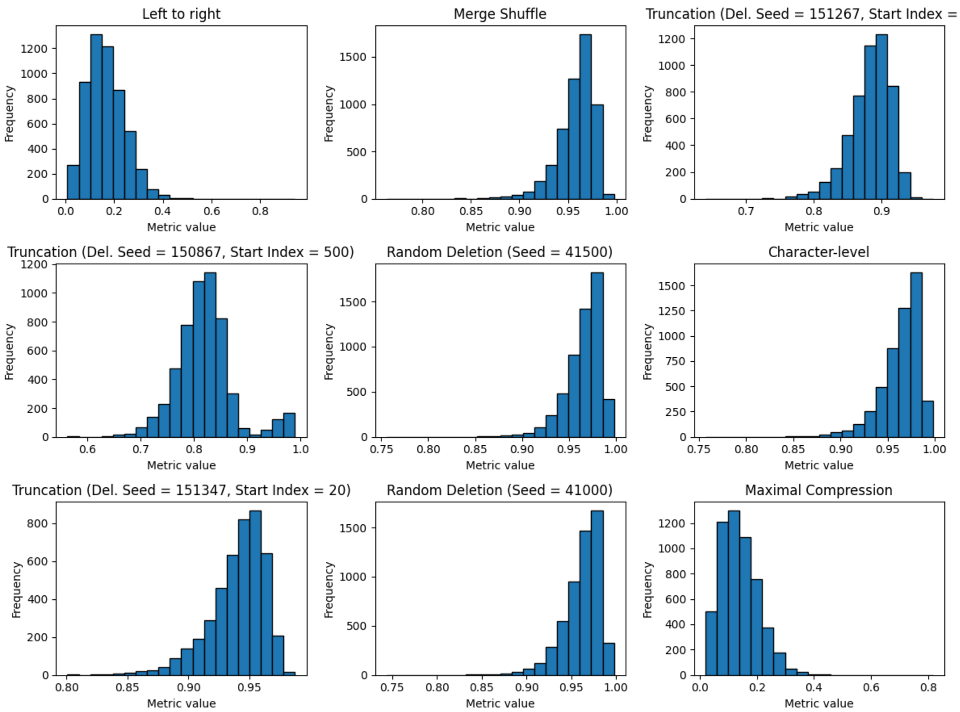

Figure 6: Token-level Levenshtein edit distance between the tokenization of S2ORC prompts obtained from the standard tokenizer and alternative tokenizers (open-ended generation subset). Lower is closer to standard.