arXiv:2410.23684v2 [cs.CL] 10 Oct 2025

Improbable Bigrams Expose Vulnerabilities of Incomplete Tokens in Byte-Level Tokenizers

Eugene Jang[1*] **Kimin Lee**[2] **Jin-Woo Chung**[3] **Keuntae Park**[3] **Seungwon Shin**[2]
[1]Northeastern University [2]KAIST [3]S2W Inc.
[1]jang.eu@northeastern.edu [2]{kiminlee,claude}@kaist.ac.kr
[3]{jwchung,keuntae.park}@s2w.inc

Abstract

Tokenization is a crucial step that bridges human-readable text with model-readable discrete tokens. However, recent studies have revealed that tokenizers can be exploited to elicit unwanted model behaviors. In this work, we investigate *incomplete tokens*, i.e., undecodable tokens with stray bytes resulting from byte-level byte-pair encoding (BPE) tokenization. We hypothesize that such tokens are heavily reliant on their adjacent tokens and are fragile when paired with unfamiliar tokens. To demonstrate this vulnerability, we introduce *improbable bigrams*: out-of-distribution combinations of incomplete tokens designed to exploit their dependency. Our experiments show that improbable bigrams are significantly prone to hallucinatory behaviors. Surprisingly, the same phrases have drastically lower rates of hallucination (90% reduction in Llama3.1) when an alternative tokenization is used. We caution against the potential vulnerabilities introduced by byte-level BPE tokenizers, which may introduce blind spots to language models.

1 Introduction

Tokenization is an important step in the large language model (LLM) pipeline, serving as the bridge between text inputs and the discrete tokens processed by the model. Improper tokenization can lead to undesirable behaviors, such as glitch token hallucinations (Rumbelow and Watkins, 2023; Land and Bartolo, 2024) and errors in numerical reasoning (Singh and Strouse, 2024). Furthermore, it has been observed that tokenizers can introduce elements of unfairness (Petrov et al., 2023) and bias (Ovalle et al., 2024) into models. Given the increasing duration and cost of model training, investigating and understanding the impact of the tokenizer is becoming increasingly important.

Recent works specifically analyze the tokenization step to identify inputs that provoke unwanted model behaviors. Land and Bartolo (2024) propose embedding layer heuristics to identify *glitch* tokens — undertrained tokens that cause hallucinations and also enable jailbreaks (Geiping et al., 2024). Wang et al. (2024) create adversarial questions designed to induce incorrect segmentation by the tokenizer, which degrades model performance.

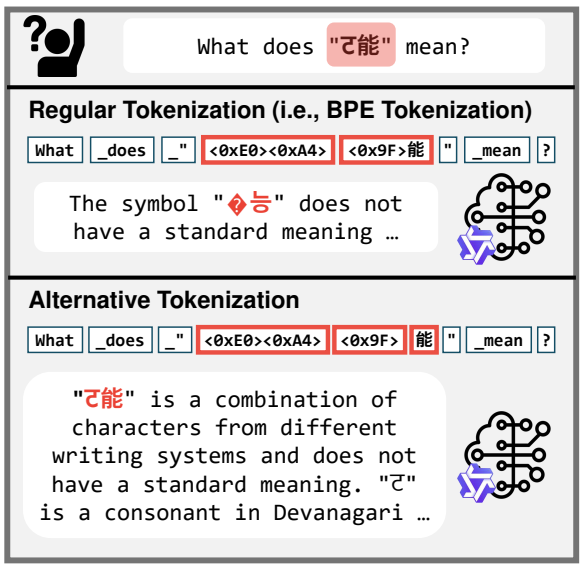


Figure 1: An improbable bigram phrase that combines two incomplete tokens to cause hallucinatory behaviors in the Qwen2.5 model. This behavior persists across multiple models and with well-trained tokens. An alternative tokenization of the same phrase does not cause hallucinations.

In this work, we investigate a specific vulnerability associated with *incomplete tokens* in byte-level byte pair encoding (BPE) tokenizers. These tokens, also known as undecodable tokens, are byte-level tokens that cannot be decoded independently and must appear in conjunction with certain other tokens to form legal Unicode characters. We explore the fragility of these tokens by constructing *improbable bigrams*, which are unlikely yet permissible combinations of two incomplete tokens. Similar to glitch tokens, improbable bigrams cause hallucinations to benign user requests, as seen in Figure 1.

[*] Work performed while at S2W Inc.

Tokenizer	Vocab Size	Incomplete Tokens	Incomplete Bigrams
Meta-Llama-3.1	128k	1224	71k
Exaone-3.0	102k	1222	36k
Qwen2.5	151k	1320	39k
Mistral-Nemo	131k	1307	135k
Command-R-v01	255k	2956	1479k

Table 1: Vocabulary sizes and number of incomplete tokens in different tokenizers. Incomplete bigrams refer to the total number of legal bigrams that can be created by combining two incomplete tokens.

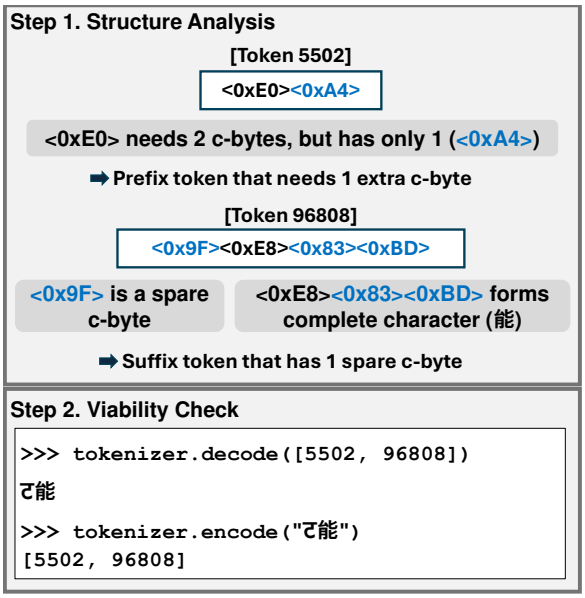


Figure 2: Incomplete tokens are analyzed for their structure to find potential bigram candidates. C-bytes (continuation bytes) are colored in blue. Tokens with compatible structures are tested for viability and reachability through an decode-encode check.

However, unlike glitch tokens, improbable bigrams can cause hallucinatory behavior even when their constituent tokens are well-trained.

Our experiments across multiple LLM families (such as Llama-3.1, Qwen2.5, and Mistral-Nemo) demonstrate that these bigrams are significantly more prone to producing hallucinations compared to bigrams formed from complete tokens. Intriguingly, alternative tokenizations of the same phrases rarely exhibit hallucinations. We believe that our findings contribute to the growing body of research on understanding the potential weaknesses of BPE tokenization, with implications for the development of more robust language models.

2 BPE Tokenization

Byte pair encoding (BPE), originally developed as a data compression algorithm (Gage, 1994), has evolved into a popular tokenization scheme used by many modern language models (Sennrich et al., 2016). BPE tokenization iteratively expands the vocabulary by selecting frequent pairs of existing tokens as new vocabulary items. In particular, byte-level BPE, which applies the BPE algorithm at the byte-level rather than the character level (Wang et al., 2019), offers distinct advantages. Firstly, byte-level tokenization eliminates out-of-vocabulary issues by representing all Unicode characters (154,998 in Unicode v16.0) as combinations of 256 base bytes, ensuring comprehensive coverage. Secondly, it allows for more efficient data compression. The widespread adoption of byte-level tokenization in cutting-edge models such as GPT-4 and Llama 3.1 underscores its effectiveness.

While some have attributed BPE's effectiveness to its compression (Gallé, 2019; Goldman et al., 2024), recent work has challenged this assumption (Schmidt et al., 2024). Bostrom and Durrett (2020) point out that greedy compression prioritizes frequency over linguistically meaningful boundaries. This observation is supported by other works that report model improvements from morphologically motivated modifications to BPE tokenizers (Limisiewicz et al., 2024; Bauwens and Delobelle, 2024). These works demonstrate that incorrect segmentation by BPE tokenization cause suboptimal model performance.

3 Incomplete Tokens and Improbable Bigrams

Byte-level BPE tokenizers are agnostic not only to morphemes but also to character boundaries. Training with the BPE algorithm on texts where characters are encoded with multiple bytes (e.g., Cyrillic (2 bytes), CJK Unified Ideographs (3 bytes)) often results in the formation of *incomplete tokens*. Incomplete tokens contain stray bytes (like <0x9F> in the token '<0x9F>能') that do not correspond to recognizable characters. These incomplete tokens constitute a small but significant portion of the vocabulary in byte-level models. Table 1 presents the number of discovered incomplete tokens in the vocabularies of various models employing byte-level tokenizers (see Appendix A for details).

While incomplete tokens are indistinguishable from complete tokens from the model's perspective

Task	User prompt
Repeat	`Repeat this phrase exactly: "{Phrase}"`
Definition	`What does "{Phrase}" mean?`
Do You Know?	`Today I heard about "{Phrase}". Do you know what this means?`

Usernames	`I am trying to learn python. What is the output of this? Print the output only with no further explanation.`
	`Usernames = ['LiquidBoxi胖胖', '{Phrase}', 'ɹəʞɹəd', 'Jhocam(≠ ˋɜˊ )', 'AR皮球']`
	`print(Usernames[0:2])`

Table 2: Prompt templates used to test phrase-level hallucinations of target phrases. At test time, **{Phrase}** is replaced with the tested phrase. If the model fails to repeat the phrase in *all* four prompts, we consider the phrase to be hallucinatory.

during training, their structure can influence how they are trained in subtle ways. First, incomplete tokens can only occur alongside other incomplete or single-byte tokens, as the "stray bytes" of incomplete tokens require additional bytes to form a viable character. This restriction limits the range of tokens they can be associated with. Second, the potential characters that stray bytes can form into may have little to no semantic overlap, leading to a representation that is inherently ambiguous and heavily reliant on neighboring tokens for disambiguation. We hypothesize that these factors exacerbate one another, causing incomplete tokens to become overly dependent on their context and brittle in response to unfamiliar adjacent tokens.

To demonstrate this vulnerability, we create phrases that exploit incomplete tokens to cause hallucinations. First, we analyze the structure of incomplete tokens, categorizing those that end with stray bytes as *prefixes* and those that begin with stray bytes as *suffixes*. We further label prefixes and suffixes by the number of stray bytes they require to form a complete character, to find potential bigram combinations. We create adversarial combinations of these tokens, termed *improbable bigrams*, as illustrated in Figure 2. These bigrams exploit the dependency of two incomplete tokens to complete each other. We note that not all combinations are viable due to tokenization rules and merge priorities. Table 1 lists the number of all possible legal bigrams created from incomplete tokens across different tokenizers. To identify bigrams that are particularly out-of-distribution, we use the heuristic of multilinguality. When a bigram forms a string, we check the Unicode script type of each character. If the string contains characters from different Unicode scripts, we assume it is highly unlikely to have been encountered during training. In our experiments, we demonstrate that *improbable bigrams* are prone to hallucinations.

Models	Improbable Bigrams	Baseline Bigrams
Llama 3.1	48/100 (48%)	0/100 (0%)
Exaone	77/100 (77%)	20/100 (20%)
Qwen2.5	33/100 (33%)	0/100 (0%)
Mistral-Nemo	52/71 (73%)	1/71 (1%)
Command-R	49/100 (49%)	8/100 (8%)

Table 3: Number of hallucinations by **improbable bigram** phrases (with incomplete tokens) and by **baseline bigram** phrases (with complete tokens).

4 Hallucination Experiments

Models We experiment using five recently released instruction-trained models that use byte-level BPE tokenization: Meta-Llama-3.1-8B-Instruct[1], EXAONE-3.0.-7.8B-Instruct[2], Qwen2.5-32B-Instruct[3], Mistral-Nemo-Instruct-2407[4], and C4AI-Command-R-v01[5]. We use greedy decoding for all experiments.

Evaluation Prompts We assess each model's ability to accurately process prompts containing improbable bigram phrases. We use four prompt templates designed to induce the model to repeat a target phrase from the input, regardless of the phrase's nonsensical or unusual nature, as detailed in Table 2. Our goal is to find phrases that models cannot process for any prompt. Therefore, we

[1]https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

[2]https://huggingface.co/LGAI-EXAONE/EXAONE-3.0-7.8B-Instruct

[3]https://huggingface.co/Qwen/Qwen2.5-32B-Instruct

[4]https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407

[5]https://huggingface.co/CohereForAI/c4ai-command-r-v01

consider a phrase to have induced hallucinations only if the model fails to correctly repeat it across all four prompt templates.

Incomplete Token Selection For each model, we construct improbable bigrams using the procedure outlined in Section 3. Noting that undertrained tokens often lead to hallucinations, we focus our experiments on well-trained tokens to isolate confounding factors. To do this, we employ the embedding heuristics in (Land and Bartolo, 2024) to detect whether a token in the vocabulary is undertrained. Specifically, the heuristics estimate the degree of training for each token by taking L2 norms and cosine distances in the embedding matrix. We use these metrics to order the entire vocabulary, and discard the lower (undertrained) half from our experiments. From the upper half (well-trained half) of the vocabulary, we identify the incomplete tokens and create up to 100 improbable bigrams for each model. Analysis of the languages used in the improbable bigrams (Appendix B) show language-pair distributions differ widely between models.

Baselines To better demonstrate the fragility of improbable bigrams, we create a baseline of bigrams constructed with complete tokens. For each improbable bigram, we create a complete token counterpart by replacing the prefix and suffix with similarly-trained tokens. That is, if the initial incomplete token is the *i*-th token in the ranked vocabulary, we find a complete token that is slightly less trained (e.g. the (*i-1*)-th token). This method ensures that each improbable bigram is compared to a baseline of complete bigrams with a similar level of undertraining.

Results As shown in Table 3, improbable bigrams suffer from significantly higher rates of hallucination than their complete token counterparts across all tested models. This is distinct from hallucinations involving glitch tokens, as previously investigated by Land and Bartolo (2024), which was attributed to undertrained tokens. These results suggest that even well-trained incomplete tokens can struggle to faithfully represent textual inputs.

5 Alternative Tokenization

Our previous experiments demonstrate that improbable bigram phrases are difficult for models to process. We conduct further experiments to attribute this difficulty to the incomplete tokens. We repeat the experiments, but pre-segment the target phrase to avoid character boundary-crossing tokenization.

Models	Incomplete Token Hallucinations	Alternative Tokenization Hallucinations
Llama 3.1	0.48	0.05 (↓90%)
Exaone	0.77	0.50 (↓35%)
Qwen2.5	0.33	0.12 (↓64%)
Mistral-Nemo	0.73	0.01 (↓98%)
Command-R	0.49	0.55

Table 4: Frequency of hallucinations from improbable bigram phrases with original tokenization (**Incomplete Tokens**) and presegmented tokenization (**Alternative Tokenization**). Reduction of hallucinations by alternative tokenization is denoted by ↓ when applicable.

For instance, the improbable bigram phrase "サーミ能" is normally tokenized into incomplete tokens "サー<0xE3><0x83>" and "<0x9F>能". However, it can be presegmented to isolate the character formed from stray bytes ("ミ"). We split the string into three parts ("サー", "ミ", and "能") and tokenize each separately. Afterwards, the tokens can be appended together to create an alternative token representation of the same phrase.

Compared to improbable bigrams, the alternative tokenization sequences can be suboptimal in two important ways. First, these sequences cannot be generated by the tokenizer, ensuring that they are out-of-distribution for the model. Second, the sequences consist of more than two tokens, requiring the model to recall more tokens correctly to repeat the phrase accurately.

Table 4 shows results of the experiment repeated using the alternative tokenization scheme. Despite the suboptimalities of alternative tokenization, most models, with the exception of Command-R (discussed in Appendix B) show significantly higher performance when alternative tokenization is used. The same phrases tend to hallucinate significantly less when alternative tokenization is used, suggesting that the hallucinatory behaviors are being caused by the incomplete tokens.

6 Discussion

Risks of Unrepeatability We have used a set of repetition-inducing prompts as a simple task that identifies phrases that cause model failures. The repetition failures, stemming from incomplete tokens, are possibly indicative of other robustness failures linked to incomplete tokens. However, in certain scenarios, non-repeatability itself can cause real world harms. We highlight few such scenarios.

- Reliable Code and Data Handling: When models interact with code or databases, they must correctly preserve variable names or fixed values. If models are unable to preserve certain phrases, they can compromise the integrity of data.

- Adversarial unrepeatability: Adversaries may exploit unrepeatable phrases to avoid intervention from LLM agents. For instance, an attacker can set their username to an unrepeatable phrases to evade a moderator agent.

- Model Fingerprinting: Since improbable bigrams are model-specific due to vocabulary design, they could be used to fingerprint the architecture behind a closed or anonymized LLM service. This can facilitate sophisticated targeted attacks on private LLMs (e.g. adversarial triggers optimized for the target architecture). This can also create implications for model evaluations that rely on model anonymity (e.g. Chatbot Arena).

Circumvention Strategies Our findings suggest that model developers should consider the consequences of vulnerable tokenizer design. The vulnerability of incomplete tokens can be circumvented by preventing incomplete tokens from entering the final tokenizer vocabulary (before model training begins). One method could involve vocabulary pruning (Bauwens and Delobelle, 2024) after tokenizer training to remove incomplete tokens. Another method would be to constrain BPE merges to respect character boundaries during tokenizer training (Land and Arnett, 2025). If full Unicode coverage is not deemed necessary for a certain model, developers may opt to choose tokenization at the character-level before tokenizer-training. We hope that demonstrating potential vulnerabilities of the default byte-level BPE can motivate more careful design of tokenizers.

7 Conclusion

Through improbable bigrams, we demonstrate vulnerabilities of incomplete tokens present in byte-level BPE tokenizers. We conclude that incomplete tokens are significantly more prone to hallucinatory behaviors compared to complete tokens. Our findings suggest that model developers must be mindful of potential unwanted behaviors caused by incomplete tokens.

Limitations

This work provides evidence to suggest the fragility of incomplete tokens. We assess phrase-level hallucinations as a proxy for larger potential harms of model misbehavior. We limit our investigations to phrase-level hallucinations, rather than factual hallucinations. While we occasionally observe that the models confidently assert incorrect explanations about what is likely non-existent terminology, this was not possible to evaluate systematically. One reason is that the tested phrases contain scripts spanning many languages beyond the expertise of the authors. Another is that the baseline levels of factual hallucination when asked about *ordinary* terms in low-resource languages are already very high. In pilot experiments, we found that even models with high conversational fluency often make mistakes in explaining non-existent terms.

Our findings on improbable bigrams causing hallucinations were consistent in all 5 models. However, our secondary experiment on alternate tokenization did not cause improvements for the Cohere Command-R model. It is difficult to isolate which design choices of the model contributes to this effect. One hypothesis is that the model's significantly larger vocabulary size, which is approximately double the other studied models, may have impacted token training. We have also conducted an analysis on the languages of the incomplete bigrams as a possible explanation, but found it inconclusive. A more detailed investigation is left to future work. There is a possibility that these findings may be influenced by other unknown and undocumented dependencies relating to tokenizer parameters, training parameters, and training data.

References

Thomas Bauwens and Pieter Delobelle. 2024. BPE-knockout: Pruning pre-existing BPE tokenisers with backwards-compatible morphological semi-supervision. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5810–5832, Mexico City, Mexico. Association for Computational Linguistics.

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.

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

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. In *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 1375–1381, Hong Kong, China. Association for Computational Linguistics.

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. 2024. 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 and virtual meeting. Association for Computational Linguistics.

Sander Land and Catherine Arnett. 2025. Bpe stays on script: Structured encoding for robust multilingual pretokenization. *Preprint*, arXiv:2505.24689.

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

Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. 2024. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15059–15076, Bangkok, Thailand. Association for Computational Linguistics.

Anaelia Ovalle, Ninareh Mehrabi, Palash Goyal, Jwala Dhamala, Kai-Wei Chang, Richard Zemel, Aram Galstyan, Yuval Pinter, and Rahul Gupta. 2024. Tokenization matters: Navigating data-scarce tokenization for gender inclusive language technologies. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Mexico City, Mexico. Association for Computational Linguistics.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*.

Jessica Rumbelow and M Watkins. 2023. Solidgoldmagikarp (plus, prompt generation). 2023. *URL: https://www. alignmentforum. org/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-promptgeneration*.

Craig W. Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization is more than compression. *Preprint*, arXiv:2402.18376.

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.

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

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2019. Neural machine translation with byte-level subwords. *ArXiv*, abs/1909.03341.

Dixuan Wang, Yanda Li, Junyuan Jiang, Zepeng Ding, Guochao Jiang, Jiaqing Liang, and Deqing Yang. 2024. Tokenization matters! degrading large language models through challenging their tokenization. *Preprint*, arXiv:2405.17067.

A Incomplete Bigram Construction

UTF-8 characters have a flexible number of bytes. Each multi-byte character comprises of a starting byte and continuation bytes. Starting bytes indicate how many bytes the character is composed of. For example, the starting byte of a 3-byte character implies the following two characters will be continuation bytes. A more detailed explanation of the UTF-8 protocol and byte structures can be found in (Limisiewicz et al., 2024).

Using starting bytes and continuation bytes, we can identify whether a byte sequence requires more continuation bytes to be fulfilled or has excess continuation bytes. As illustrated in Figure 2, each incomplete token can be analyzed for their structure. We test all possible combinations of incomplete bigrams by pairing bigrams of complementary structures. We focus only on bigrams that users could use to induce the intended incomplete bigram structure. Note that the resulting character of conjoined bytes may not always correspond to a usable Unicode character. There is also a possibility that the resulting phrase is tokenized into a different sequence of tokens. We test this by performing a decode-encode test to ensure the resulting string is valid and indeed tokenizes to the intended prefix and suffix.

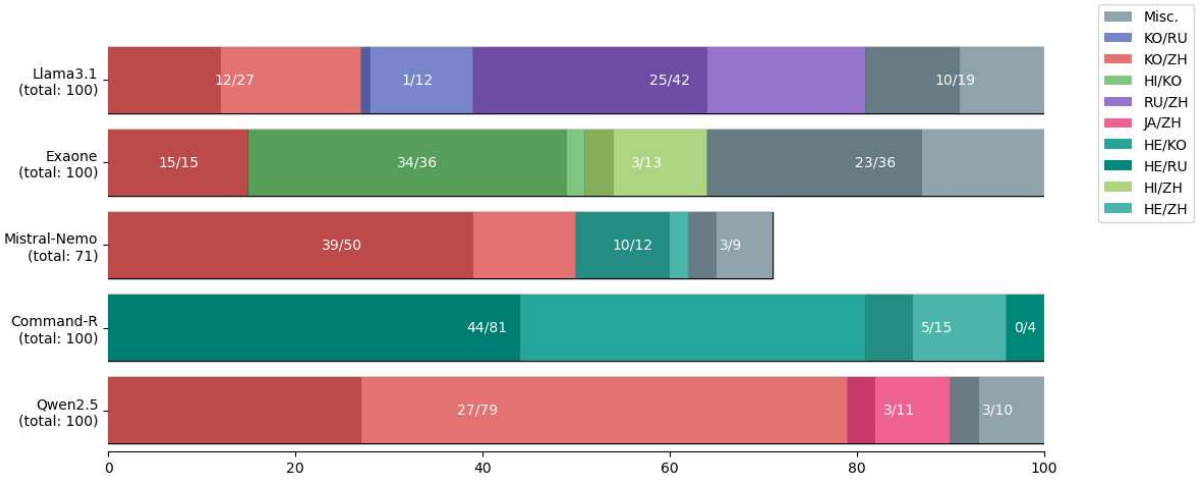
B Role of Languages in Improbable Bigrams

We analyze the scripts of the improbable bigrams used in the experiments. Note that the language distributions of the tested improbable bigrams are not indicative of all possible improbable bigrams

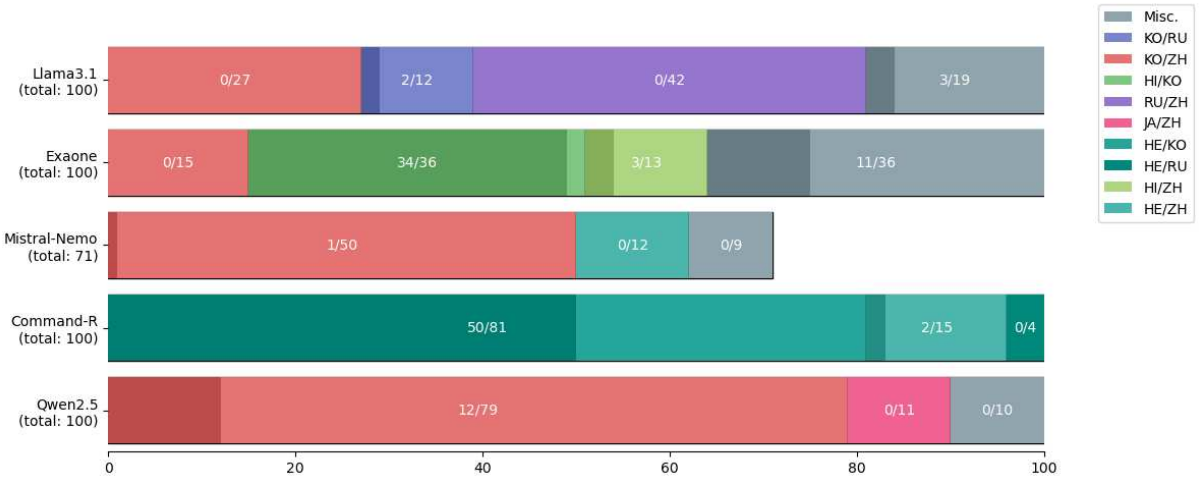
in each model's vocabulary, since the bigrams from the experiments were selected from the top half of the vocabulary when ordering by token trainedness (Section 4). Figure 3 shows the distribution of languages of the improbable bigrams for each model. We observe that improbable bigrams can occur in a wide variety of languages, with higher resource multi-byte scripts such as Chinese, Korean, and Russian being the most frequent. The diversity of languages of hallucinations suggest there is no significant influence caused by languages.

Models had different distributions of language-pairs, which is influenced by both tokenizer training and model training. For instance, Exaone had 17 unique language-pairs while Command-R had only 3 unique language-pairs. Command-R's much larger tokenizer vocabulary may have impacted the incomplete token selection. All of the 100 bigrams for Command-R contained Hebrew characters. For comparison, there are no tokens in the top-half of Llama3.1's vocabulary that resolve to Hebrew characters. Command-R is also the only model analyzed with bigrams of the Hebrew/Korean language-pair. Bigrams of this language-pair constitute a majority of the tested improbable bigrams for Command-R, and performed worse when using alternative tokenization.

Alternative Tokenization with Hebrew/Korean The Command-R model's improbable bigrams did not show improvements when using alternative tokenization. Figure 3 shows that alternative tokenization harmed performance for the 81 Hebrew/Korean bigrams. We investigate if the Hebrew/Korean language-pair is particularly prone to hallucinations when using alternative tokenization. We collect a total of 38 possible bigrams of this language combination from the Llama3.1 vocabulary. Repeating the experiments with Llama3.1's 38 Hebrew/Korean bigram pairs, using alternative tokenization reduced hallucinations (20/38) compared to regular tokenization (32/38). These results suggest that Command-R's unusual result with alternative tokenization was not caused by the language-pairing of its improbable bigrams.



(a) Improbable Bigrams



(b) Improbable Bigrams (alternative tokenization)

Figure 3: Language-pair distributions of the improbable bigrams used in experiments. Darkened colors indicate improbable bigrams that cause hallucinations.