

Journey Across the Tokenization Wasteland: An Audit of Vocabulary Hygiene in LLM Tokenizers

Suhas Pai
Fruitless Labs
research@fruitless.ai

Xiaojun Ren
Fruitless Labs
research@fruitless.ai

Abstract

Tokenization remains one of the least revisited components of large language models. Most models today still rely on static vocabularies learned by algorithms driven largely by corpus statistics; these vocabularies are then reused throughout training and inference, often even across generations of model families. In this paper, we conduct a tokenizer audit across 9 tokenizers associated with 23 LLMs, carrying out a comprehensive analysis of their vocabularies. We develop a taxonomy of vocabulary-hygiene issues, namely junk tokens, staircase tokens, unreachable tokens, undertrained (or glitch) tokens, and low-utility long-tail tokens. Across the evaluated tokenizers, we find that vocabulary-hygiene issues are widespread, but the specific failure modes are heterogeneous in nature. Each tokenizer exhibits its own characteristic weaknesses as a function of its training data and preprocessing, the algorithm used, merge dynamics, vocabulary size, and language distribution. We demonstrate that these failures are not merely cosmetic defects and that they have significant implications for cost, efficiency, multilingual representation, and the allocation of finite vocabulary capacity. Finally, we propose mitigation strategies tailored to each class of vocabulary-hygiene issue. Our results suggest that tokenizer training and vocabulary construction should be treated as an auditable and improvable part of the language model pipeline rather than as fixed infrastructure.

1 Introduction

Contemporary Large Language Models (LLMs) typically utilize static subword vocabularies constructed from algorithms such as BPE (Byte Pair Encoding) [27], WordPiece [25], and UnigramLM [6], and widely used implementations such as SentencePiece [7] and tiktoken [13]. These methods are preferred for their simplicity, speed, and language-agnostic nature. However, their vocabulary construction objectives are based on corpus-level statistics and compression heuristics rather than downstream tasks. As a result, these subword vocabularies exhibit artifacts from their training corpus. Our work studies these artifacts and their implications for downstream tasks.

In this paper, we audit nine modern tokenizer vocabularies associated with 23 different LLMs. We introduce a taxonomy of vocabulary-hygiene issues. Junk tokens are entries whose usage is concentrated in low-quality web artifacts such as SEO (Search Engine Optimization) spam, gambling pages, adult-content link farms, boilerplate navigation text, and e-commerce residue. Staircase tokens are intermediate merge residues produced during greedy vocabulary construction. Unreachable tokens exist in the vocabulary but cannot be emitted under normal encoding rules. Undertrained or glitch tokens receive little or no training signal during language-model pretraining. Low-utility long-tail tokens consume vocabulary capacity while contributing little to ordinary compression.

Our strongest empirical finding is the concentration of Mandarin junk tokens in OpenAI tokenizer vocabularies, especially o200k_base. We find 35% of all multi-character Mandarin tokens are associated with gambling, pornography, lottery, and SEO-spam domains. In contrast, vocabularies from DeepSeek and Qwen tokenizers contain far fewer such entries.

We further show that these artifacts have practical consequences: on generic Mandarin text, DeepSeek tokenizes more efficiently than OpenAI’s modern tokenizer, while the pattern reverses on gambling and adult-content text. Thus, vocabulary capacity appears to have been allocated in a way that improves compression for polluted web domains while worsening token parity for ordinary Mandarin usage.

This paper makes several contributions to the auditing of tokenization schemes and their resulting vocabularies. First, we identify junk Mandarin tokens in OpenAI’s o200k_base, cl100k_base, and o200k_harmony vocabularies, and study their implications for multilingual token parity. Second, we investigate the origins of these tokens using experiments on a representative subset of C4 (Colossal Clean Crawled Corpus) before and after data filtering, producing a candidate junk-token dataset that we map back to modern LLM tokenizers. Third, we show that many of these tokens appear consistent with black-hat SEO and spam patterns associated with Chinese illegal online gambling and adult-content promotion. Fourth, we introduce a taxonomy of vocabulary-hygiene issues: junk, staircase, unreachable, undertrained, and low-utility long-tail tokens, and empirically audit these issues across modern tokenizers. Finally, we connect vocabulary hygiene to inference cost, multilingual representation, robustness, representational geometry, and finite vocabulary capacity, and propose mitigation strategies for each class of issues.

2 Related Work

Several works in the literature have investigated the impact of tokenizer choice on model efficiency and behavior. There have been studies on evaluating tokenizers based on compression, fertility [14], multilingual coverage, downstream performance, and robustness. Goldman et al. [12] demonstrate correlations between compression and downstream performance and argue that compression is an intrinsic metric for tokenizer quality. Reddy et al. [23] show that there are diminishing returns to increasing tokenizer training data size, and attribute this to the pre-tokenization process. Similar work by Velayuthan et al. [24] shows that pre-tokenization has a bigger impact on vocabulary construction than the tokenization algorithm itself, as they bound the maximum token length that can be formed for a given pre-token. Our work builds on this line of research by moving beyond aggregate metrics to a fine-grained audit of individual vocabulary entries. Specifically, we investigate whether vocabulary capacity is allocated to tokens that are reachable, well-trained, linguistically useful, and representative of expected usage.

The closest prior study to our analysis on semantic pollution is by Zhang et al. [18], which investigated data pollution in Chinese tokens in OpenAI’s o200k_base vocabulary, identifying a large number of tokens associated with pornography, gambling, and related low-quality SEO content. Their work shows that tokenizer vocabularies contain artifacts from the tokenizer training data and thus can provide clues about its composition and data distribution. However, we note many limitations in their work in terms of false positives and negatives. Their token classification fails to identify tokens that look superficially innocuous when their literal meaning is taken, but in fact represent gambling or porn site names. Meanwhile, tokens with sexual or sensitive meanings are tagged as polluted tokens, even though they may have legitimate uses.

Another related line of work investigates vocabulary artifacts created by BPE-style algorithms. Sun et al. [22] coin the phrase intermediate merge residues to describe tokens that are present in the vocabulary as an artifact of the merge process, but are rarely emitted during model generation because later merges absorb them into longer tokens. They introduce Litetoken, a method for identifying and removing such residue tokens, while preserving model performance. Our work is closely related, but differs in its treatment of what counts as residue. We note that certain merge artifacts can still play compositional roles as productive stems, and introduce the concept of staircase tokens, which are a chain of tokens in which progressively longer tokens absorb most occurrences of their prefixes, while distinguishing between productive and unproductive staircases by considering their marginal compression contribution.

Undertrained tokens, also called glitch tokens, were widely investigated after the discovery of tokens like SolidGoldMagikarp [21], which induced anomalous behavior in GPT-2 and GPT-3. This was followed by a systematic analysis of these types of tokens by Land et al. [8], who define them as tokens that appear in the tokenizer training data and consequently its vocabulary, but are nearly or entirely absent from the language model’s training data. Our work incorporates their analysis but places it within a broader taxonomy of vocabulary-hygiene issues, by distinguishing them from polluted tokens which correspond to tokens that are low-quality artifacts but are still well-trained due to their presence in the model training corpora. By measuring both categories, we can test differences between the tokenizer training data composition and the model training data composition.

3 Vocabulary-Hygiene Issues

We conducted a tokenizer audit across 9 tokenizers associated with 23 LLMs, namely OpenAI’s o200k_base, Google’s Gemini 2.5 Pro local tokenizer, DeepSeekV3-0324, Qwen3-8B, GLM-4.5-Air, ByteDance Seed-OSS-36B, LLaDA-8B-Instruct, Mistral NeMo Instruct, and Phi-4-miniInstruct. For historical and junk-token comparisons, we also examine related variants including OpenAI GPT-2, r50k_base, p50k_base, p50k_edit, cl100k_base, o200k_harmony, DeepSeek variants, Qwen variants, and Mistral variants. Our audit included systematic analysis of each of the tokens in the vocabularies being analyzed. We found a large number of tokens providing very little end user utility but consuming valuable vocabulary capacity. Based on the type of issue found, we developed a taxonomy of vocabulary-hygiene issues, categorizing each such problematic token into junk tokens, staircase tokens, unreachable tokens, undertrained tokens, and low-utility long-tail tokens. We will now discuss each of these in detail.

3.1 Junk Tokens

We define junk tokens as vocabulary entries whose presence in the tokenizer training corpora is concentrated in low-quality web artifacts but is typically not encountered in ordinary model usage. Examples of junk tokens include tokens representing SEO spam, gambling sites, lottery advertisements, adult-content link farms, web navigation text, and e-commerce boilerplate. While a large proportion of such junk tokens have Adult/NSFW themes, tokens with sexual or sensitive meanings are not inherently considered as junk tokens according to our definition as they have legitimate uses. We do not aim to pass moral judgement on token identities.

We distinguish junk tokens from glitch tokens and long-tail tokens. Glitch tokens are undertrained because they are scarcely present in the model training data, while junk tokens are usually well-trained because of their high rate of incidence in the model training set. Long-tail tokens appear in lower frequencies in the training set but they still have legitimate uses, like being a scientific or domain-specific term, or a word/phrase in an underrepresented language.

The effect of junk tokens might be amplified due to the incentives of large-scale pre-training. Recent work from Mohri et al. [11] shows that there might be a bitter lesson for data filtering; in high-compute settings, large models sometimes benefit from low-quality data, thus reducing the need for rigorous data filtering during pretraining, but incidentally introducing junk tokens in the vocabulary.

Mandarin Junk Tokens in OpenAI: The most surprising observation from our audit is the large concentration of Mandarin junk tokens in OpenAI’s o200k_base tokenizer vocabulary, which is used by models such as GPT-5.5 and 5.4. The vocabulary of o200k_base contains a total of 7,454 Mandarin tokens, 1,478 or 20% of which have been classified as junk tokens. The proportion becomes substantially higher when we remove mono-character tokens from the statistics, since they do not represent sufficient context or meaning. We found that 35% of all Mandarin tokens containing two or more characters have been classified as junk tokens. In comparison, DeepSeek’s vocabulary has a total of 35,199 Mandarin tokens with only 8 or 0.02% classified as junk tokens. None of the other tokenizers had a Mandarin junk token

Category	Tokens	% of Mandarin junk
Gambling / lottery SEO	845	57.2%
Adult / NSFW link farms	421	28.5%
Video / content-farm residue	85	5.8%
SEO / boilerplate	56	3.8%
Scraped-web residue	41	2.8%
Games / e-commerce residue	30	2.0%
Total	1,478	100.0%

Table 1: Manually verified Mandarin junk-token categories in `o200k_base`. Counts are grouped into coarse categories to emphasize that the issue is not ordinary sensitive language, but vocabulary capacity allocated to SEO-spam and low-quality web artifacts.

rate over 0.1%, showing that this issue is specific to OpenAI’s tokenizers. Table 1 shows the category breakdown for `o200k_base`.

Mandarin Junk Token Identification Methodology: We first identified Mandarin tokens in each vocabulary using Unicode range filtering. Starting with tokens containing characters from the CJK (Chinese Japanese Korean) Unicode ranges, only tokens that include Han characters or Bopomofo symbols were retained. Each token is then classified using a web agent that searches the Web to infer its semantic origin and usage context. The tokens are classified into one of 29 topic-based categories. The final classification results are drawn from the union of our classifier and Zhang et al.’s [18] polluted Chinese tokens classifier.

Label validation: The classifier labels were treated only as candidates. For `o200k_base`, the classifier identified 1,612 candidate Mandarin junk tokens. Using manual verification, we rejected 181 false positives, and a second pass over the remaining Mandarin tokens added 47 missed junk tokens, yielding 1,478 final labels. A token was labeled as junk only when its dominant usage was concentrated in low-quality web artifacts such as gambling pages, adult-content link farms, lottery promotion, SEO boilerplate, or web navigation artifacts. Tokens with sexual, sensitive, geographic, medical, literary, or domain-specific meanings were not labeled as junk when they had ordinary uses. Thus, our labels capture corpus-provenance artifacts rather than moral or content-based undesirability.

We also repeated Land et al.’s [8] analysis on identifying undertrained tokens based on behavioral clues. None of these tokens exhibited anomalous behavior, suggesting that these tokens are not undertrained, which means they likely existed in the pre-training dataset as well.

Token Parity: One of the practical implications of junk tokens is visible in token-parity analysis. We constructed two datasets comprising of 200 Mandarin passages each: one comprised of generic passages and one comprised of gambling and adult-content SEO text. To control for passage length, in Table 2 we report mean token counts per 1,000 Unicode characters. On generic Mandarin, the DeepSeek tokenizer uses 29.9% fewer tokens than OpenAI’s `o200k`-based tokenizer. This is unsurprising as DeepSeek’s vocabulary consists of 4.7x more Mandarin tokens than GPT-5.5’s.

However, the situation is reversed on text drawn from porn or gambling domains, where `o200k_base` requires 20.0% fewer tokens on average to represent the same information in comparison to DeepSeek-v4-pro. The same tokenizer that is relatively less efficient in Mandarin is unusually efficient when it comes to representing text from porn or gambling domains. The vocabulary capacity has been allocated in a way such that low-quality web domains are favored over normal language use.

Thus, junk tokens are not cosmetic defects, but in fact represent an opportunity cost and a misallocation of representational bandwidth. This has implications for inference cost, multilingual parity, and model robustness.

Embedding Geometry Distortion: The presence of junk tokens in a vocabulary is not only an efficiency concern. It also has consequences for the geometry of the embedding space. A token string’s representation is shaped by the contexts in which it appears during training.

Dataset	N	OpenAI	DeepSeek	Tokenizer advantage
Generic Mandarin	200	731.4 $\pm$ 84.3	512.6 $\pm$ 61.7	DeepSeek: 29.9% fewer
Gambling/adult SEO Mandarin	200	397.2 $\pm$ 73.1	496.5 $\pm$ 89.4	OpenAI: 20.0% fewer

Table 2: Token-parity comparison between OpenAI’s o200k-based tokenizer and the DeepSeek tokenizer. Values are mean token counts per 1,000 Unicode characters, with standard deviations over 200 passages.

When its occurrences are dominated by polluted web text, a text-embedding model may associate that string more strongly towards the junk domains than with its literal or surface meaning.

We observe this effect when using OpenAI text embeddings to encode decoded vocabulary strings. Since we do not have access to the input-embedding matrix from OpenAI’s GPT models, we cannot make a claim that the learned token embeddings of models using o200k_base are distorted. Instead, we measure the geometry of the vector space from an OpenAI text-embedding model over the decoded strings corresponding to vocabulary entries. In this embedding space, Mandarin junk tokens do not scatter as undertrained tokens would. They form a dense, well-separated cluster, which we visualize in Figure 1. This shows that the embedding model associates these strings with a polluted-web semantic domain, rather than as unrelated rare strings.

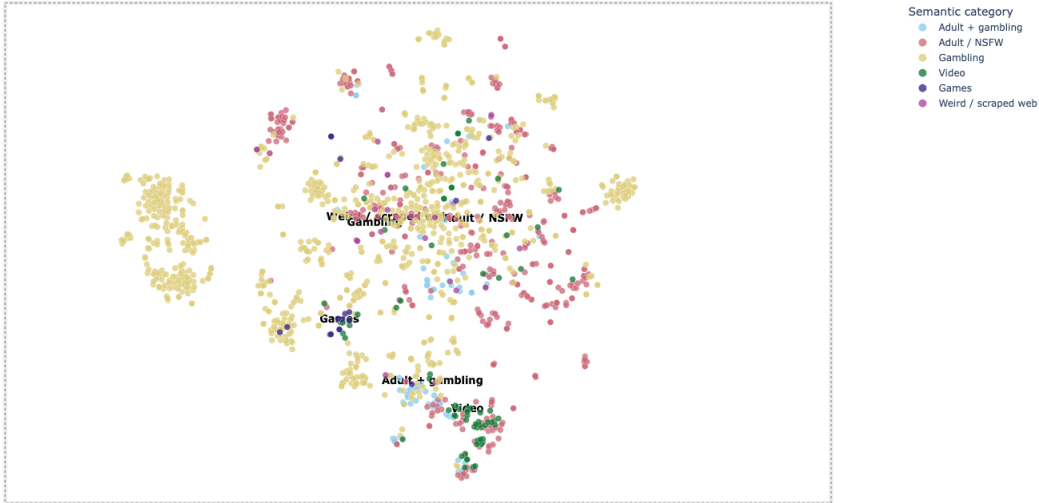


Figure 1: t-SNE diagram of Mandarin junk tokens in o200k_base, cl100k_base, and o200k_harmony

The downstream effect is semantic pollution in the decoded-token embedding space. Because a polluted token’s geometry reflects the company it kept during training, a token whose literal meaning is innocuous can sit closer in embedding space to terms from the polluted domain than to its true semantic neighbors. In our analysis, ”九州”, whose literal meaning is the Chinese name of the island Kyushu in Japan, appears closer to gambling-related strings relative to other Japanese place names due to it also being the name of a Taiwanese syndicate managing underground money laundering and illegal gambling networks. We interpret this as evidence that decoded vocabulary entries can carry domain provenance.

Investigating Black Hat SEO: During our investigation, we attempted to find the source of Mandarin junk tokens. Yang et al. [26] identified more than 967K illegal Chinese gambling websites that employ black hat SEO techniques. Their work reveals the sheer volume of promotional strategies used by illegal gambling websites. Du et al. reveal that in addition to

keyword stuffing, black hat SEO techniques also inject inbound links to reputable websites by using bots to post spam links in popular forums, compromising reputable sites with link injections, purchasing link exchange or reciprocal linking services, and leveraging expired domains with high page ranks to construct link networks [2].

New black hat SEO techniques are more sophisticated and target crawlers directly. One of the strategies utilized is to set up a “Spider Pool” that uses wildcard DNS to generate a large number of subdomains from a single domain, creating a dense network of spam pages designed to attract and trap crawlers [2]. This results in the crawler spending a disproportionate amount of resources on spam content and index manipulated search results containing porn, illegal gambling and lottery, gaming, and e-commerce. While we do not have access to OpenAI’s tokenizer-training dataset or its crawler behavior, the Mandarin junk tokens we observe are consistent with this kind of large-scale SEO spam, suggesting a potential lack of sufficient data filtering before training the tokenizer and the language models. We investigated the presence of junk tokens in languages beyond Mandarin and were not able to find a similarly large class of SEO-style junk tokens, suggesting that the Mandarin case reflects a specific interaction between web-scale tokenizer training and a high-volume Chinese spam ecosystem.

Identifying Junk Tokens Beyond Mandarin: To identify multilingual junk tokens beyond Mandarin, we constructed a junk token candidate dataset from the C4 corpus. We first ran a filtering pipeline over the unclean version of C4 to remove web artifacts, yielding a cleaned corpus and a complementary set of rejected text. We then trained separate BPE vocabularies on each: one on the original corpus and one on the text that the filter removed. Tokens that appear in the rejected-text vocabulary but not in the original vocabulary are treated as candidate junk tokens, on the assumption that artifacts surface as distinct vocabulary entries only once clean text no longer dominates the merge statistics. Finally, we discard extremely low-frequency candidates, since these are more likely to be legitimate rare tokens than systematic web artifacts. The full algorithm can be found in Appendix A.

Across modern tokenizers, candidate junk token matches range from 0.80% to 3.01% of the vocabulary, with the full table provided in Appendix B.

4 Staircase Tokens

Staircase tokens, or intermediate residues, refer to tokens that were added to the vocabulary during the tokenizer training process but were merged into longer tokens later on during training. This is more prevalent in tokenizer algorithms like BPE, where the vocabulary is constructed through a sequence of greedy frequency-based merges.

Most tokenizers use BPE-style algorithms, where merges are based on frequency statistics. When a long frequent string is added, the shorter intermediate sub-strings needed to construct it may remain in the vocabulary even if they are rarely useful after later merges absorb them. Since the merges happen at the character level, they happen in a step-by-step manner, leading to a staircase fashion. We call these *Staircase Tokens*. Note in Table 3, “精益求精” is a common four-character idiom in Chinese; however, its staircased components “精益” and “精益求精” have no obvious meaning or usage.

Tokenizer	Type	Step	Token ID	Token
qwen3_8b	prefix_right_extension	0	118102	精益
qwen3_8b	prefix_right_extension	1	118103	精益求精
qwen3_8b	prefix_right_extension	2	118104	精益求精

Table 3: Example of a prefix right extension staircase token ladder in qwen3_8b.

We conducted an experiment across 9 tokenizers to determine the number of staircase tokens that exist in each vocabulary. Table 4 shows the intermediate residue tokens in each tokenizer, including the number of staircase tokens.

Table 4: Staircase token counts (prefix and suffix ladder) per vocabulary.

Tokenizer	Provider	Prefix	Suffix	Total	% vocab
openai_o200k_base	OpenAI	18	6	24	0.01%
gemini_2_5_pro_local	Google	112	108	220	0.08%
deepseek_v3_0324	DeepSeek	8	10	18	0.01%
qwen3_8b	Alibaba/Qwen	6,128	20	6,148	4.05%
glm_4_5_air	Zhipu/Z.ai	30	25	55	0.04%
bytedance_seed_oss_36b	ByteDance Seed	14	19	33	0.02%
llada_8b_instruct	LLaDA	10	6	16	0.01%
mistral_nemo_instruct	Mistral	16	16	32	0.02%
phi_4_mini_instruct	Microsoft	28	18	46	0.02%

We observe thousands of such staircase tokens in the Qwen tokenizer vocabulary, with the earlier tokens in the staircase almost never used. These tokens are not found in other tokenizer vocabularies that were part of our study. We consider a staircase token as residue if it a) has low independent emission frequency, and b) has low contextual diversity, meaning their surrounding tokens in a representative corpus are similar.

Not every intermediate fragment can be dismissed as residue. Sun et al. [22] identify English stems such as 'delet' and 'includ' as intermediate residue. However, these tokens are stems of English words because they represent the prefix of common terms with similar meanings and follow Porter Stemmer conventions [17].

5 Unreachable Tokens

An unreachable token is a token that exists in the vocabulary but cannot be generated by the model when using the tokenization algorithm. Unreachable tokens are the clearest form of token waste, as they consume vocabulary capacity but can never be used in practice. In BPE-style tokenizers, one way in which a token can become unreachable is when merge priority prevents a token from ever being selected for generation because other merge rules always take precedence. In tokenization schemes containing pre-tokenization, a token may cross a pre-token boundary and thus be never generated during inference.

We performed a reachability test across 9 tokenizers. To perform a reachability test, we decode each vocabulary token into its string representation. We then re-encode the string in isolation and check whether the encoded token appears in the output.

Table 5: Unreachable token counts and proportions across tokenizer vocabularies.

Tokenizer	Provider	Total	Unreachable	% vocab
openai_o200k_base	OpenAI	200,000	307	0.15%
gemini_2_5_pro_local	Google	262,144	318	0.12%
deepseek_v3_0324	DeepSeek	128,815	338	0.26%
qwen3_8b	Alibaba/Qwen	151,669	316	0.21%
glm_4_5_air	Zhipu/Z.ai	151,365	350	0.23%
bytedance_seed_oss_36b	ByteDance Seed	155,121	316	0.20%
llada_8b_instruct	LLaDA	126,349	375	0.30%
mistral_nemo_instruct	Mistral	131,072	272	0.21%
phi_4_mini_instruct	Microsoft	200,029	307	0.15%

We note that there are a few unreachable tokens across all tokenizers but they are not substantial in number. Yet, they still represent valuable vocabulary capacity that is being wasted. Unreachable tokens exhibit similar behavior as glitch tokens. This is because unreachable tokens are never emitted during the pre-training process and thus receive no training signal. Unreachable tokens are a tokenizer-level property, while glitch tokens are a model-level property. Unlike other forms of undesirable tokens, unreachable tokens are straightforward to identify and also straightforward to remove or replace.

6 Undertrained Tokens

We define undertrained or glitch tokens as tokens that exist in the tokenizer vocabulary but receive little to no training signal during model pretraining. Such tokens induce anomalous model behavior and thus negatively impact downstream tasks. Glitch tokens appear when there is a mismatch between tokenizer training data and language model pretraining data, specifically when tokens that have relatively higher frequencies in the tokenizer training data barely appear if not completely absent from language model pretraining data, due to differences in training corpora, or preprocessing and filtering steps.

In order to identify glitch tokens, we perform behavioral and embedding-level tests prescribed by Land et al. [8]. While most glitch token research has been performed in the context of English and Mandarin tokens, we run the analysis across the entire vocabularies of the tokenizers we cover. From our analysis, we found 127 glitch tokens in o200k_base and 37 glitch tokens in the DeepSeek tokenizer.

7 Idiosyncratic Long-Tail Tokens

Tokenizer scaling laws [4] state that our current vocabulary sizes are smaller than optimal. Therefore, there has been a trend of recent language models having larger vocabularies. However, the new tokens added to the vocabulary typically tend to be rare long-tail tokens rather than tokens that would actually reduce the amount of tokens needed to represent text.

Whether a token is considered a long-tail token or not is subjective, even if we have access to pretraining corpus statistics. We developed a pseudo-measure to detect the number of long-tail tokens in a vocabulary by identifying tokens that are not present in any of the other compared vocabularies, excluding tokens from non-English languages. This proxy is not definitive, as a token absent from other vocabularies may reflect legitimate domain or language coverage. Therefore, we use this measure only for discovering candidates which then merit further inspection.

Table 6: Long-Tail Tokens

Tokenizer	Total Tokens	Long-Tail Tokens	Long-Tail %
gemini_2_5_pro_local	262,144	90,707	34.62%
bytedance_seed_oss_36b	155,121	23,633	15.35%
mistral_nemo_instruct	131,072	14,811	11.39%
llada_8b_instruct	126,349	14,399	11.46%
qwen3_8b	151,669	11,158	7.42%
deepseek_v3_0324	128,815	5,088	3.98%
glm_4_5_air	151,365	2,908	1.93%
openai_o200k_base	200,000	1,562	0.78%
phi_4_mini_instruct	200,029	225	0.11%

8 Mitigation Strategies

Different vocabulary-hygiene failures are best addressed by different mitigation techniques. Unreachable tokens can be detected mechanically and removed or replaced. Staircase tokens can be reduced by modifying the tokenizer training algorithm. PickyBPE [1], for example, removes intermediate residue tokens during tokenizer construction. AdaptBPE [9] replaces low-utility tokens with high-frequency tokens from a target domain corpus. In this vein, several works investigate modifying vocabularies for new languages or domains including Haliuk et al. [3], Remy et al. [19], Remy et al. [20], Minixhofer et al. [10].

Preventing junk tokens from entering the vocabulary requires better tokenizer-training data filtering and vocabulary audits before tokenizer release. Because junk tokens may be well-trained, they cannot be identified simply by looking for anomalous behavior. They require

semantic and corpus-provenance analysis. Long-tail token bloat can be addressed by usage-aware vocabulary sizing, as rare tokens are not inherently bad, but additional vocabulary entries should be justified by compression, coverage, or downstream utility.

A more radical solution to vocabulary-hygiene issues is to replace static subword vocabularies with dynamic vocabularies or even tokenization-free approaches. BLT (Byte Latent Transformer)[16] operates on byte sequences that are grouped into dynamically sized patches. FLEXITOKENS introduces byte-level segmentation that can enable adaptation to new domains and languages[15]. H-Net truly removes tokenization as a separate step in the language model pipeline, introducing an end-to-end hierarchical architecture that learns content and context dependent segmentation[5]. Our vocabulary-hygiene studies provide empirical motivation for this direction, as dynamic tokenization or tokenization-free approaches are a promising way to ensure language models allocate representation capacity according to the needs of downstream tasks rather than on corpus statistics.

9 Conclusion

We treated the tokenizer vocabulary as an auditable artifact and audited 9 tokenizers associated with 23 LLMs, introducing a taxonomy of vocabulary-hygiene issues: junk, staircase, unreachable, undertrained, and long-tail tokens. These issues are pervasive but heterogeneous across tokenizers, with each tokenizer carrying the fingerprint of its training corpus, algorithm, vocabulary size, and language distribution.

Our most consequential finding concerns junk tokens. Their concentration in OpenAI’s o200k_base vocabulary demonstrates how vocabulary capacity has been spent encoding low-quality web artifacts traceable to black hat SEO content. This is not cosmetic: it distorts token parity and degrades representational geometry. Our analysis is limited to public tokenizer vocabularies and decoded-token string embeddings; it does not reconstruct proprietary training corpora or directly measure proprietary model input embeddings.

Vocabulary construction therefore deserves the same scrutiny as any other stage of the language modeling pipeline. The appropriate repair technique for vocabulary-hygiene issues depends on the failure mode. Staircase tokens and unreachable tokens can often be removed mechanically, while junk tokens require better upstream filtering and semantic audits, undertrained tokens require better corpus alignment, and long-tail bloat requires usage-aware vocabulary sizing. More broadly, the persistence of these issues strengthens the case for dynamic and tokenization-free architectures, where capacity is allocated to downstream needs rather than the statistics of an unfiltered crawl.

References

- [1] Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. BPE gets picky: Efficient vocabulary refinement during tokenizer training, 2024. URL <https://arxiv.org/abs/2409.04599>.
- [2] Kun Du, Hao Yang, Zhou Li, Haixin Duan, and Kehuan Zhang. The ever-changing labyrinth: A large-scale analysis of wildcard DNS powered blackhat SEO. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pp. 245–262, Austin, TX, 2016. USENIX Association. URL <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/du>.
- [3] Mykola Haltiuk and Aleksander Smyńwiński-Pohl. Model-aware tokenizer transfer, 2025. URL <https://arxiv.org/abs/2510.21954>.
- [4] Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling, 2025. URL <https://arxiv.org/abs/2501.16975>. Accepted at ICML 2025.
- [5] Sukjun Hwang, Brandon Wang, and Albert Gu. Dynamic chunking for end-to-end hierarchical sequence modeling, 2025. URL <https://arxiv.org/abs/2507.07955>.
- [6] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018. URL <https://arxiv.org/abs/1804.10959>.
- [7] Taku Kudo and John Richardson. 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*, pp. 66–71, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL <https://aclanthology.org/D18-2012>.
- [8] Sander Land and Max Bartolo. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP 2024)*, pp. 11632–11648. Association for Computational Linguistics, 2024. URL <https://aclanthology.org/2024.emnlp-main.649.pdf>.
- [9] Vijini Liyanage and François Yvon. AdaptBPE: From general purpose to specialized tokenizers, 2026. URL <https://arxiv.org/abs/2601.21665>.
- [10] Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3992–4006, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.293. URL <https://aclanthology.org/2022.naacl-main.293>.
- [11] Christopher Mohri. A bitter lesson for data filtering, 2026. URL <https://arxiv.org/abs/2605.19407>.
- [12] Matan Eyal Kris Cao Idan Szpektor Reut Tsarfaty Omer Goldman, Avi Caciularu. Unpacking tokenization: Evaluating text compression and its correlation with model performance. *arXiv preprint arXiv:2403.06265*, June 2024. URL <https://arxiv.org/abs/2403.06265>. To appear in EMNLP 2024, Findings.
- [13] OpenAI. tiktoken: A fast BPE tokeniser for use with OpenAI’s models. <https://github.com/openai/tiktoken>, 2022. GitHub repository.
- [14] Volodymyr Ovcharov. The tokenizer tax across 25 european languages: Domain invariance, cross-lingual few-shot effects, and the ukrainian penalty, 2026. URL <https://arxiv.org/abs/2605.24718>.

- [15] Abraham Toluase Owodunni, Orevaoghene Ahia, and Sachin Kumar. FLEXITOKENS: Flexible tokenization for evolving language models, 2025. URL <https://arxiv.org/abs/2507.12720>.
- [16] Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens, 2024. URL <https://arxiv.org/abs/2412.09871>.
- [17] Martin F. Porter. An algorithm for suffix stripping. *Program: Electronic Library and Information Systems*, 14(3):130–137, 1980. doi: 10.1108/eb046814.
- [18] Haoting Qian Liu Yan Tianwei Zhang Ke Xu Qi Li Minlie Huang Hewu Li Han Qiu Qingjie Zhang, Di Wang. Speculating llms’ chinese training data pollution from their tokens. *arXiv preprint arXiv:2508.17771*, September 2025. URL <https://arxiv.org/abs/2508.17771>. v2, submitted 30 Sep 2025.
- [19] François Remy, Pieter Delobelle, Bettina Berendt, Kris Demuynck, and Thomas Demeester. Tik-to-tok: Translating language models one token at a time: An embedding initialization strategy for efficient language adaptation, 2023. URL <https://arxiv.org/abs/2310.03477>.
- [20] François Remy, Pieter Delobelle, Hayastan Avetisyan, Alfiya Khabibullina, Miryam de Lhoneux, and Thomas Demeester. Trans-tokenization and cross-lingual vocabulary transfers: Language adaptation of LLMs for low-resource NLP, 2024. URL <https://arxiv.org/abs/2408.04303>. Accepted at COLM 2024.
- [21] Jessica Rumbelow and Matthew Watkins. SolidGoldMagikarp (plus, prompt generation). LessWrong, February 2023. URL <https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation>. Accessed: 2026-06-24.
- [22] Yike Sun, Haotong Yang, Zhouchen Lin, and Muhan Zhang. LiteToken: Removing intermediate merge residues from BPE tokenizers. *arXiv preprint arXiv:2602.04706*, 2026. doi: 10.48550/arXiv.2602.04706. URL <https://arxiv.org/abs/2602.04706>.
- [23] Yuval Pinter Chris Tanner Varshini Reddy, Craig W. Schmidt. How much is enough? the diminishing returns of tokenization training data. *arXiv preprint arXiv:2502.20273*, June 2025. URL <https://arxiv.org/abs/2502.20273>. v4, submitted 16 Jun 2025.
- [24] Menan Velayuthan and Kengatharaiyer Sarveswaran. Egalitarian language representation in language models: It all begins with tokenizers. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING 2025)*, pp. 5987–5996, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL <https://aclanthology.org/2025.coling-main.400/>. Anthology ID: 2025.coling-main.400.
- [25] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. URL <https://arxiv.org/abs/1609.08144>.
- [26] Hao Yang, Kun Du, Yubao Zhang, Shuang Hao, Zhou Li, Mingxuan Liu, Haining Wang, Haixin Duan, Yazhou Shi, Xiaodong Su, Guang Liu, Zhifeng Geng, and Jianping Wu. Casino royale: A deep exploration of illegal online gambling. In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC ’19)*, pp. 500–513, San Juan, PR, USA, 2019. ACM. doi: 10.1145/3359789.3359817.

- [27] Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, 2023. URL <https://arxiv.org/abs/2306.16837>.

A Appendix A

Algorithm 1 Multilingual Junk Token Candidate Extraction

Require: Raw corpus C , artifact filter F , tokenizer training procedure T , vocabulary size V , minimum frequency threshold τ , smoothing constant ϵ

Ensure: Junk token candidate set J

- 1: Apply filter F to C to obtain cleaned corpus

$$C_{\text{clean}} = F(C)$$

and rejected corpus

$$C_{\text{rej}} = C \setminus C_{\text{clean}}.$$

- 2: Train tokenizer T on C_{clean} with vocabulary size V to obtain vocabulary

$$\mathcal{V}_{\text{clean}}.$$

- 3: Train tokenizer T on C_{rej} with vocabulary size V to obtain vocabulary

$$\mathcal{V}_{\text{rej}}.$$

- 4: Compute rejected-only candidate tokens:

$$\Delta = \mathcal{V}_{\text{rej}} \setminus \mathcal{V}_{\text{clean}}.$$

- 5: **for all** $t \in \Delta$ **do**

- 6: Compute frequency $f_{\text{rej}}(t)$ in C_{rej} .

- 7: Compute frequency $f_{\text{clean}}(t)$ in C_{clean} .

- 8: Compute enrichment score

$$e(t) = \log_2 \left(\frac{f_{\text{rej}}(t) + \epsilon}{f_{\text{clean}}(t) + \epsilon} \right).$$

- 9: **end for**

- 10: Retain only enrichment-supported candidates:

$$\Delta = \{t \in \Delta \mid e(t) > 0\}.$$

- 11: Filter low-frequency candidates:

$$J = \{t \in \Delta \mid f_{\text{rej}}(t) \geq \tau\}.$$

- 12: Rank candidates in J by decreasing enrichment score $e(t)$.

- 13: Match each candidate token in J against target tokenizer vocabularies using exact byte-string equivalence.

- 14: **return** ranked junk token candidate set J .
-

B C4 Rejected-Text Candidate Matches

Table 7: Matches to C4 rejected-text junk-token candidates across tokenizer vocabularies. These multilingual candidate counts are derived from the C4 filtering experiment and are distinct from the manually verified Mandarin junk-token labels in Section 3.1.

Tokenizer	Candidate matches	Total vocab	% vocab
<i>GPT-2 era (50k vocab)</i>			
openai_gpt2	1,265	50,257	2.52%
openai_r50k_base	1,265	50,257	2.52%
openai_p50k_base	1,270	50,281	2.53%
openai_p50k_edit	1,270	50,284	2.53%
<i>Modern tokenizers</i>			
openai_cl100k_base	1,635	100,277	1.63%
openai_o200k_base	2,328	200,019	1.16%
openai_o200k_harmony	2,328	201,088	1.16%
hf_deepseek_v3	1,849	128,815	1.43%
hf_deepseek_v3_2	1,849	128,816	1.43%
hf_qwen3	1,805	151,669	1.19%
hf_qwen3_6	1,993	248,077	0.80%
hf_mistral_v03	986	32,768	3.01%

C Tokenization Visualization Example



Figure 2: DeepSeek tokenization of a junk-token sample.



Figure 3: OpenAI tokenization of the same junk-token sample.