Understanding Secret Leakage Risks in Code LLMs: A Tokenization Perspective

Meifang Chen[†], **Zhe Yang**[‡], **Huang Nianchen**[§],
Yizhan Huang[†*], **Yichen Li**[†], **Zihan Li**[¶], and **Michael R. Lyu**[†]
[†]The Chinese University of Hong Kong [‡]Nanyang Technological University
[§]University of Southern California [¶]Fudan University
yzhuang22@cse.cuhk.edu.hk

Abstract

Code secrets are sensitive assets for software developers, and their leakage poses significant cybersecurity risks. While the rapid development of AI code assistants powered by Code Large Language Models (CLLMs), CLLMs are shown to inadvertently leak such secrets due to a notorious memorization phenomenon. This study first reveals that Byte-Pair Encoding (BPE) tokenization leads to unexpected behavior of secret memorization, which we term as *gibberish bias*. Specifically, we identified that some secrets are among the easiest for CLLMs to memorize. These secrets yield high character-level entropy, but low token-level entropy. Then, this paper supports the biased claim with numerical data. We identified that the roots of the bias are the token distribution shift between the CLLM training data and the secret data. We further discuss how gibberish bias manifests under the "larger vocabulary" trend. To conclude the paper, we discuss potential mitigation strategies and the broader implications on current tokenizer design.

1 Introduction

Code Large Language Models (CLLMs) are reshaping how software is built, maintained, and evolved, introducing AI-driven automation across every stage of the software development lifecycle. Recent academic advances (Hui et al., 2024; Hugging Face, 2024; Guo et al., 2024), demonstrate significant improvements in code generation, understanding, and reasoning across multiple programming languages and tasks. These models have been evaluated on challenging benchmarks showing state-of-the-art performance in code synthesis, completion, and bug repairs. On the commercial side, tools like Claude Code (Anthropic, 2025), Codex (OpenAI, 2026), and Cursor (Cursor, 2025) have operationalized CLLMs by integrating the technology directly into real-world development environments to provide real-time coding assistance, documentation generation, and automated testing. Such tools have achieved rapid market penetration; for instance, Claude Code has seen swift developer adoption, quickly accumulating over 111,000 cumulative npm downloads by early 2026 (Gradually AI, 2026).

However, the broad adoption of CLLMs is increasingly raising concerns in the community. Code LLMs are shown to have strong capability of *memorization* – with proper prompts, they emit training data verbatim. For instance, CLLMs may leak documentations, statements, logs, and configuration files (Yang et al., 2024a; Wu et al., 2025; Pearce et al., 2022). CLLMs may even memorize and leak code *secrets*. These secrets include API keys to online services, private keys, passwords, and URLs, posing severe security and privacy risks. The secrets appear in training corpus, since careless programmers push their secrets to code hosting services like GitHub. Therefore, with the exploding use of CLLMs, the topic is becoming important in the AI safety community. We illustrate the risk in Fig. 1.

While most memorization works focus on prompting, training paradigms, and datasets of CLLMs, this paper explores tokenization, an under-explored yet pivotal component of CLLMs. Our research is motivated by the recently discovered relationship between entropy and memorization score. The work reveals that for a token sequence from the LLM training corpus, (an estimator) of entropy is linearly related to memorization score (Huang et al., 2026). By applying the previous discovery to the memorization of secrets, this study reveals that although some gibberish-like secrets (i.e., highly-randomized strings) are *high*-entropy at the character-level, after tokenization, some secrets are encoded to *low*-entropy to-

*Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 108–119

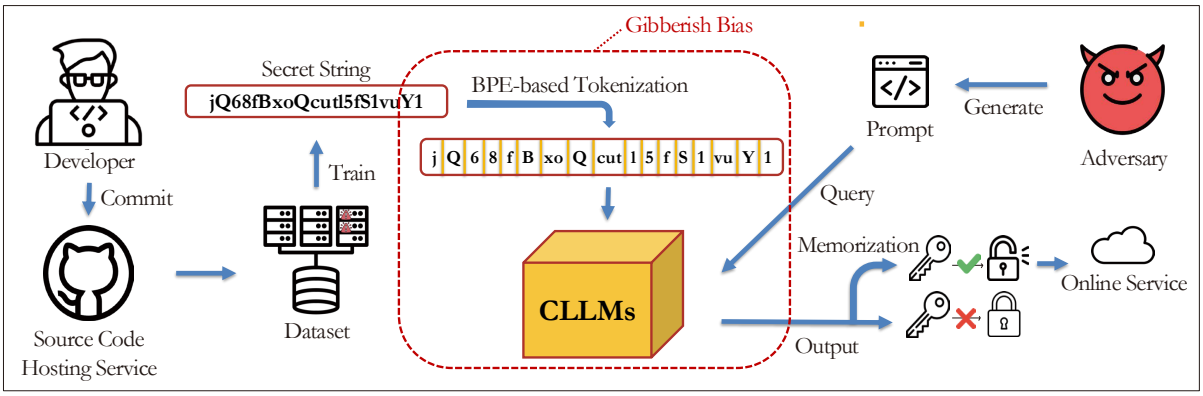

Figure 1: The risk map of secret leakage through CLLMs. The red box highlights the focus of this paper.

ken sequences, significantly reducing chances of memorization. We coin the phenomenon as **gibberish bias**. Our research results indicate that the risk of secret leakage is exacerbated with gibberish bias.

This study conducts a deeper exploration of gibberish bias. We found that gibberish bias should be attributed to Byte-pair Encoding (BPE), the most popular tokenization strategy in (Code) LLMs. We confirm that the root of gibberish bias — BPE is sensitive to distribution shift between *train* data and inference (*test*) data well. We then discuss the bias under the current trend of "larger tokenizers". The final part of the paper discusses the mitigation strategy and its broader implications for tokenizer design. The contribution of this paper is as follows:

1. This paper identifies a new risk of secret leakage through CLLMs: the tokenizer might induce gibberish bias and further exacerbate the secret leakage risk.

2. This paper explores the roots of such bias: BPE is sensitive to the distribution shift between train and test data.

3. This paper predicts that the secret leakage risk will manifest more under the current "larger tokenizer" trend.

4. The paper discusses potential mitigation strategies and broader implications for the community.

2 Background

2.1 Tokenization of Code LLMs

In NLP, early explorations use word-level tokenization (e.g., word2vec (Mikolov et al., 2013)),

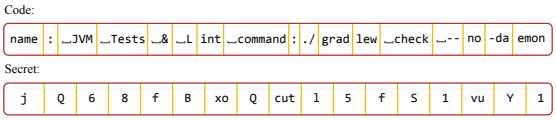

Figure 2: Use BPE to tokenize one line of normal code and secrets. Each token is a subword.

assigning each distinct word a unique index and embedding. This approach imposed a fixed vocabulary and sometimes resulted in the failure to handle out-of-vocabulary (OOV) words (Mikolov et al., 2013). Later, researchers developed subword tokenization strategies that segment words into smaller units.

The process typically includes three sequential stages: *(1) Pre-tokenization:* a preprocessing step that imposes rules on the raw text, such as whitespace splitting, normalization, or restrictions on allowable character sequences. *(2) Vocabulary Construction:* given a corpus and a target vocabulary size, an algorithm selects a set of subword units that constitute the vocabulary, Each subword is regarded as a *token*. *(3) Segmentation:* using the constructed vocabulary, this step operationalizes the mapping from raw text to subword sequences.

We note that before LLM training, model developers run stages (1) and (2) to establish the vocabulary. The vocabulary should be strictly adhered to all stages of LLM training (i.e., pre-training, post-training). During LLM training and inference, only stages (1) and (3) are applied. In decoder-only LLMs, a tokenizer assigns an input string to a sequence of numerical IDs, which are further transformed into embeddings via an embedding matrix.

The core of tokenization is the vocabulary construction algorithm (stage (2)), and the Byte-Pair

Encoding (BPE) (Sennrich et al., 2016; Gage, 1994) is the de facto standard for LLMs since GPT-2 (Radford et al., 2019). It is a greedy, data-driven merge algorithm that iteratively combines the most frequent adjacent character or byte pairs from the training corpus to build a subword vocabulary. Starting from individual symbols, BPE yields multi-character tokens for frequent patterns (e.g., "ing", "tion") while splitting rare words into smaller units. The total merging step is a hyper-parameter, and depends on the empirical decision of model trainers.

Code LLMs, including Deepseek Coder (Guo et al., 2024), StarCoder2 (Hugging Face, 2024), Qwen2.5-Coder (Hui et al., 2024), generally employ BPE-based tokenization, following the tokenizers on LLMs. For CLLM tokenizers, the vocabularies are slightly adapted towards code-related tasks. For example, many Code LLMs adapt vocabularies on codebases to improve generation fidelity and ensure compilable output (Wang et al., 2021). We further showcase how tokenization works on an example string in Fig. 2.

2.2 Code Secrets and their Entropy

Software developers need *secrets* to authenticate these third-party services as part of system integration. The secrets include API keys, access tokens, and private keys. While secrets are sensitive assets during software development, careless developers may hard-code secrets in their code and push it to online code hosting services like GitHub. Recent studies have shown that a vast amount of secrets are exposed in public software repositories (Basak et al., 2023; Feng et al., 2022; Meli et al., 2019; Zhang et al., 2023; Wu et al., 5). These secrets are collected as part of the training corpus of CLLMs; hence, they might be accidentally leaked by CLLMs.

Table 1: Common secrets and their corresponding regex patterns.

Secret type	Regex
AWS Access Key ID	`AKIA[0-9A-Z]{16}`
Google API Key	`AIza[0-9A-Za-z-_]{35}`
Tencent Cloud Secret ID	`AKID[0-9a-zA-Z]{32}`

An important property of code secrets is that secrets typically exhibit high (char-level) entropy (Shannon, 2001). For a human being, the strings look like gibberish. High entropy indicates high uniqueness. Consequently, for online service providers, entropy is a pivotal design consideration for the security of their secrets. Secrets typically follow a specific format, hence can be characterized by regular expressions. Table 1 presents examples of secrets of popular online services.

Mathematically, denote a secret as a sequence $s = (s^1, s^2, ..., s^{|s|})$, where each atomic element could be either a char or a token. Denote the set of all possible outcomes as $\mathcal{V}$. Denote the expected frequency of x as $p(x)$, the entropy H of secret s is

$$H(s) \triangleq -\sum_{x \in \mathcal{V}} p(x) \log p(x). \quad (1)$$

Taking GitHub personal access tokens [1] (Harvey, 2021) as an example, such tokens follow the format specified by the regular expression `ghp_[a-zA-Z0-9]{36}`. The char-level entropy for this token is 5.915 bits, which is close to the maximal entropy of 5.977 bits. We defer the detailed calculation to appendix B.

3 Motivating Study: BPE Is *Counterintuitive* on Secret Tokenization

Secrets are designed at character-level – the regular expression defines the secret format, and the randomized part is random *characters*. However, CLLMs process these strings on the token-level. This section then presents a motivating study using visualizations, and reveals that BPE indeed results in *counterintuitive* behaviors.

3.1 Case Study

This case study discusses the tokenization example in Figure 2. The figure illustrates how Deepseek Coder (Guo et al., 2024) tokenizes a part of normal code and a secret (substring). We observe a significant difference in **token granularity**: Although two strings present the same token count, the source code is significantly longer measured in character-level length.

For typical source code, the tokenizer chunks text in a way that is much expected: a token typically consists of multiple characters. Some words are split into shorter sub-words. For example, the word "-daemon" is represented by two tokens, "-da" and "emon". In contrast, the situation is much different for secrets. Most of the tokens are one character. Other tokens are characters of length

[1] "Tokens" in "personal access tokens" refers to the strings for authentication purposes. Readers shall not confuse with "tokens" used in (C)LLM tokenization.

Figure 3: Visualizing how Qwen2.5-Coder (left) and Deepseek Coder (right) tokenize a 2-char sub-string of secrets. Denote the substring as a_1a_2, a_1 corresponds to the horizontal coordinates of a pixel, and a_2 corresponds to the vertical coordinates of a pixel.

2 and 3. Interestingly, the tokenizer identifies an English word "cut" in a gibberish-like string as a token.

Even with one example, we could observe how BPE-based tokenization might induce gibberish bias. Tokens of secrets are distributed in a highly non-uniform way. They include chars of length 1, 2, 3, maybe $n > 10$. Tokens of shorter char-lengths may appear with higher frequency, while longer ones may be less frequent. In information theory, uniform distribution exhibits the maximal entropy possible, and hence non-uniform random variables exhibit less entropy. With less entropy, secrets are increasingly likely to be memorized. We leave the detailed calculation to Section 5.

3.2 Tokenizing 2-char secret sub-strings

To demonstrate how tokenizers perform poorly on secrets, this section shows the tokenization of a 2-char randomly-generated string. The characters are selected from a vocabulary of size 76, i.e., letters (a-z, A-Z), digits (0-9), and other symbols. These chars are commonly used by secrets. We enumerate all the combinations of a 2-char string given the vocabulary. We examine each string whether it is encoded to *one* token (in yellow) or *two* tokens (in brown) and present the result in Figure 3. There is clearly a non-random distribution of different encoding strategies. In general, we observe certain character pairs that the tokenizer tends not to split: (lowercase letter, lowercase letter), (uppercase letter, uppercase letter), (symbol, symbol), (letter, symbol). However, as one easily finds out, many non-trivial corner cases exist. The overall decision boundary is thus governed by a complex interplay of statistical heuristics. Similar patterns are observed on the Deepseek Coder tokenizer. We further observe consistent patterns on unigram-based tokenizers (XLNet, T5); see Appendix D.

4 Gibberish Bias

Section 3.1 demonstrated that on secrets, BPE-based tokenization may not function as ideally as on normal data. In other words, it is potentially biased – we term it **gibberish bias**. This section aims to formally describe gibberish bias using mathematical notations. Inspired by several research works on secret memorization, we characterize gibberish bias by memorization. Secret leakage has strong implications for the community, since upon leakage, these credentials may grant an adversary access to online services, bringing severe cybersecurity concerns.

4.1 Preliminaries

A recent LLM memorization work (Huang et al., 2026) has shown that entropy, the essential property that comes with the design of code secrets, is closely related to memorization. The work discovers the so-called *Entropy-Memorization Law*, which could be formally described as follows.

Denote a fixed pre-trained LLM θ, prompt (a token sequence) p, the golden answer (a token sequence) s, and a memorization score measuring the difference between LLM continuation and the golden answer $d(\theta(p), s)$.

The work studies two metrics, **entropy** and **normalized entropy** of the answer sequence, $M(s)$, and $\overline{M}(s)$. Note that the entropy notation is slightly twisted from $H(s)$ in Equation 1. The subtle difference lies in the outcome space $\mathcal{V}$. Since LLMs process strings at the token-level, $\mathcal{V}$ is defined over tokens. Moreover, in the Entropy-Memorization Law, the authors adopted a level-set-based calculation for $\mathcal{V}$, which we skip due to space limitations.

With the established entropy notation, normalized entropy quantifies how closely the entropy of a sequence approaches the maximum possible entropy. Normalized entropy eliminates the effects of sample space size. It is defined as

$$\overline{M}(s) \triangleq \frac{M(s)}{M_{\max}(s)} = \frac{M(s)}{\log|\mathcal{V}|}. \tag{2}$$

Under specific conditions, a loose statement of the Entropy-Memorization Law is:

1. $M(s)$ is a proxy of $d(\theta(p), s)$. The relation is positively linear.
2. $\overline{M}(s)$ is a proxy of $d(\theta(p), s)$. The relation is negatively linear.

In other words, higher entropy of a token sequence indicates a lower chance of memorization. Higher normalized entropy of a token sequence indicates a higher chance of memorization.

In this work, we are interested in studying the following metrics of the secret strings: character-level entropy $M(s)$, token-level entropy $H(s)$, and the normalized entropy, and their normalized version $\overline{M}(s)$, $\overline{H}(s)$.

4.2 Tokenizing secrets

Experimental Setup. We adopt the same experimental setup as (Huang et al., 2026). We use OLMo-1B (Groeneveld et al., 2024), a fully-open LLM with its training corpus Dolma (Soldaini et al., 2024). In the experiments, we sampled 240k sequences from Dolma. The dataset is licensed by ODC-BY, granting free access for research purposes. We reproduce the results with the same algorithm adopted in (Huang et al., 2026). We study the zero-distance set (where memorization score is 0, or "perfect memorization") and try to identify secret strings.

Experimental Results. Among all 847 instances within the zero-distance set, we identified 102 gibberish-like secrets through manual labeling. Some examples are shown below.

```
1 ESVXlO3url***Dw23KjZ
2 PszYzqs83S***N2N0Rj
3 5laXo6c1Ib***stxU9
```

Listing 1: Three examples of gibberish generated by OLMo-1B. Part of these secrets are masked due to privacy concerns.

We conduct analysis over four sets: gibberish, non-gibberish, zero-distance set, and non-gibberish in the zero distance set [2]. We adopt the entropy estimator and normalized entropy introduced in this section over these four sets. The results of our analysis are summarized in the following table.

[2]To clarify, non-secrets refers to the complement set of *labeled* secrets. Therefore, "non-secrets" may include unlabeled secrets. Due to the large size of the sampled corpus, labeling on such an extensive scale is not feasible at this stage of work.

The experimental results reveal the key findings: **high *character*-level entropy does not necessarily imply high *token*-level entropy.** In fact, at the token-level, these secrets have significantly lower entropy (8.084) than non-secrets (11.175); while at the char-level, these secrets have significantly higher entropy (6.086) than non-secrets (4.744). Besides, if we calculate the difference delta between secrets and non-secrets, there is a significant gap in normalized entropy between token-level ($\Delta = 0.806 - 0.719 = 0.087$) and character-level ($\Delta = 0.974 - 0.361 = 0.613$).

It is observed that at the token-level, the 102 identified gibberish uses more than 1k different tokens. On the contrary, at char-level, they are composed of 76 unique chars (i.e., A-Z, a-z, 0-9, and 14 other chars). That showcases the outcome space size discrepancy between char-level and token-level, and explains our finding.

The findings may deviate from our human intuition. A wrong logic chain of a human is: by the Entropy-Memorization Law, secret strings are highly randomized; hence, they have high entropy and are hard to memorize. The problem lies in the "high entropy property" naturally assumed by our human beings – human beings perceive secrets at the character level. In contrast, for LLMs, the entropy should be calculated over the token level. After tokenization, some high character-level normalized entropy strings are transformed into low entropy ones. Hence, the EM-Law suggests they should be easier to memorize than an average non-gibberish text.

We conclude this section with the full description of gibberish bias:

Gibberish bias

BPE-based tokenization transforms some of the **high *character*-level** entropy sequences into **low *token*-level** entropy sequences.

5 How Does Tokenization Induce Gibberish Bias?

The above results reveal an essential defect induced by tokenization. The findings raise an intriguing and important question: How does this happen? To explore the question, this section then discusses the design issues of BPE.

Table 2: Statistics of secret memorization at token-level (T) and char-level (C).

	Unique Elements		Entropy		Normalized Entropy	
	T	C	T	C	T	C
Zero-distance Set	3,661	105	7.834	5.110	0.662	0.761
Secrets	1,047	76	8.084	6.086	0.806	0.974
Non-Secrets	47,945	9,006	11.175	4.744	0.719	0.361
Non-Secrets in Zero-distance Set	2,897	105	7.329	4.966	0.637	0.740

Experimental Setup The following experiments include tokenizers of three representative Code LLMs: Deepseek Coder (Guo et al., 2024), Qwen2.5-Coder (Hui et al., 2024), and StarCoder2 (Hugging Face, 2024). All model weights and tokenizers are free to access for research purposes. The experiments involve comparisons on two datasets: the "secret" dataset as introduced in Section 4 and the subsample Stack V2 dataset (Hugging Face, 2024). Stack V2 is the training corpus of StarCoder2.

LLM trainers typically construct a sample corpus that shares the distribution with the training data to construct the vocabulary. Moreover, tokenizer design is fixed during the whole LLM training (and inference) process. However, in terms of distribution, secret strings are essentially different from the (Code) LLM training corpus. We regard that gibberish bias may stem from the distribution shift between secrets and a typical (Code) LLM training corpus. This also matches our intuition – secret data are highly randomized, while the CLLM training corpus typically includes source code collected from online code hosting services. Then this section aims to answer the question: *Compared with the token distribution of the code dataset to train CLLMs, how different is the distribution of tokens in secrets?*

Following the approach in the case study (Sec. 3.1), we conduct tokenization on the secret dataset and analyze the token composition of secrets. To be specific, we report the frequency of tokens based on their character length (n) for all secret-related tokens in the dataset in Fig. 4. We observed an exponential decrease in token frequency as character length n increases, as evident from the log-scale y-axis. The overall distribution is long-tailed, leading to the low entropy of the token sequence. Interestingly, we even found tokens with lengths of $n \geq 11$ characters, indicating the complexity of the tokenization decision boundary.

Ideally, the token distribution of secrets should be *uniform*; but the observed distribution of BPE tokens is *long-tailed*. From information theory, we know that uniform distribution achieves the maximal entropy (Shannon, 2001), while non-uniform distribution exhibits less entropy. The discrepancy of token distributions supports the gibberish-bias claim.

Next, we are interested in the causes of the identified long-tail distribution. How do these long char-length tokens form? We further sort the tokens on the secret dataset by frequency, and compare them with the frequency of the same token on a general CLLM dataset.

Fig. 5 ranks the most frequent 150 tokens on the secret dataset, and compares the frequency of each token on the subsampled Stack V2 dataset. For both datasets, we employ the same tokenizer StarCoder2. The tokens distribution of the secret dataset (yellow line), exhibits a much steeper and more consistent drop-off in frequency for less common tokens compared to the Stack V2 (brown line). This observed divergence in log-proportion frequencies strongly suggests a distribution shift between the two datasets, particularly for tokens beyond the most frequent few. Setting the Stack V2 as the reference data distribution, we further report the KL divergence between the secret dataset and the Stack V2. The resulting KL divergence is 2.668, demonstrating the discrepancy.

Using a CLLM tokenizer, the token distribution of secret data is long-tailed – and such a long-tailed distribution explains the low entropy of secrets, hence explains the claimed gibberish bias. Further investigation confirms the significant shift in distribution between secret data and general CLLM training data.

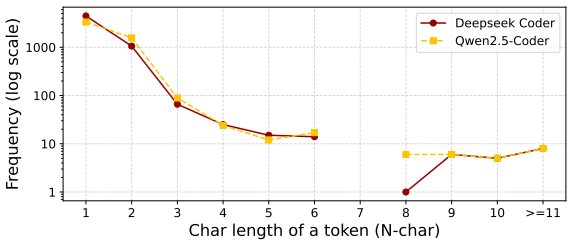

Figure 4: The distribution of n-char tokens.

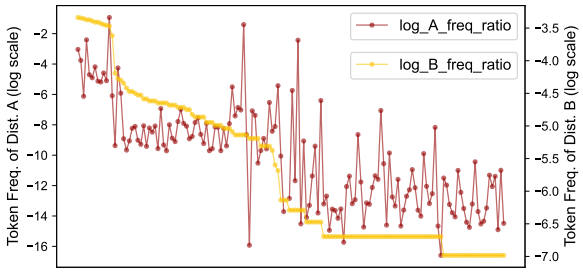

Figure 5: Token distributions on the subsampled Stack V2 dataset (Distribution A, in brown) and the secret dataset (Distribution B, in yellow).

6 Gibberish Bias Under a Larger Tokenizer Vocabulary

There is a growing consensus in both academia and industry that larger models shall benefit from a larger vocabulary (Tao et al., 2024; Huang et al., 2025). As the prevailing trends of building larger Code LLMs, it is believed that CLLMs deserve large vocabularies. Tao et al. (2024)'s work reveals that model parameters N_{nv} and the corresponding optimal vocabulary size N_v^{opt} approximately follow a *power* law. The authors further find that almost all state-of-the-art general-purpose LLMs and Code LLMs use vocab sizes smaller than the predicted optimal one. Then a question naturally arises: Will a larger vocabulary size induce more gibberish bias?

We build three new tokenizers of StarCoder2 (3B, 7B, and 15B) using the sub-sampled stack v2. We follow the *IsoFLOPs* (Tao et al., 2024) approach to generate tokenizers in "optimal" vocabulary size. IsoFLOPs analysis suggests that size 39367 for the 3B model, size 62280 for the 7B model, and size 93987 for the 15B model. We then trained these tokenizers using the subsampled Stack V2 dataset, based on the suggested size. For clarity of presentations, these tokenizers are named *SC-3B-o*, *SC-7B-o*, and *SC-15B-o*, respectively.

With three new tokenizers, we investigate gibberish bias on the secret dataset. Following (Huang et al., 2026), we report the entropy and normalized-entropy of these secrets introduced in Section 2.2 at the token-level.

To compare metrics on secrets and non-secrets, we use a "sec./non-sec." ratio. Table 3 shows the overall result. First, the results on new tokenizers converge with those of OLMo tokenizers discussed in Section 4. Compared to non-secrets, secrets exhibit lower entropy at the token level and higher entropy at the char level; secrets achieve almost maximal normalized entropy. Second, regarding the sec./non-sec. ratio, it is observed that the relative difference between secrets and non-secrets is slightly expanding. For both entropy and normalized entropy metrics, the sec./non-sec. ratio keeps deviating from the baseline ratio 1. We summarize our findings in this section as follows:

> Gibberish bias tends to be more pronounced in models employing larger tokenizer vocabularies.

7 Related Work

Empirical quantification of secret memorization. Initial memorization research focused on empirically demonstrating and quantifying this risk (Carlini et al., 2021). HCR (Huang et al., 2024a) is a method proposed to test and validate the leakage of hard-coded credentials from neural code completion tools using prompts derived from public GitHub files. Similarly, Yang et al. (2024b) conducted a systematic study showing that CLLMs memorize a broad range of data related to code. A study (Wu et al., 2025) investigates how memorization leads to general code cloning, raising concerns about copyright infringement and bug propagation. Other studies explore code clone as implementations of memorization (Ciniselli et al., 2022; Al-Kaswan and Izadi, 2023). Recent research probes the underlying mechanisms at a finer granularity. For example, DESEC (Nie et al., 2025) addresses the problem from the decoding stage.

Tokenization shapes LLM performance. Tokenizer design has been shown to cause unexpected downstream harms, such as cross-lingual unfairness (Petrov et al., 2023) and the misalignment

Table 3: Entropy and normalized entropy of secrets when using different tokenizers. "Sec." refers to "Secrets".

	Entropy				Normalized Entropy			
	SC-3B-o	*SC-7B-o*	*SC-15B-o*	*Char*	*SC-3B-o*	*SC-7B-o*	*SC-15B-o*	*Char*
Sec.	6.931	7.076	7.145	6.086	0.777	0.774	0.773	0.974
Non-Sec.	9.136	9.355	9.575	4.745	0.721	0.714	0.708	0.361
Sec./Non-Sec.	0.759	0.756	0.746	1.283	1.079	1.085	1.092	2.695

between subword tokenization and code grammar (Li et al., 2025); we identify an analogous privacy harm, showing how standard tokenizers can inadvertently reduce the token-level entropy of secrets, making them inherently more vulnerable to memorization.

8 Discussions

8.1 Implications on Safeguarding Secrets

Mitigating gibberish bias. We propose two mitigation strategies. The first is to enforce character-wise tokenization for secrets (details in Appendix C). The second, *gibberish-token elimination*, directly targets the underlying mechanism and offers better computational efficiency. It consists of two stages: (1) *identify* gibberish tokens in the tokenizer vocabulary, and (2) *eliminate* them from the CLLM. The motivation is that certain tokens are used predominantly when tokenizing gibberish yet rarely appear in natural language or code; for instance, in Qwen2.5-Coder, "aksi" (id: 37679) and "Ġll" (id: 1369) do not belong to any normal programming language. We attribute such tokens to gibberish in the tokenizer's (and LLM's) training data.

Stage 1: identifying gibberish tokens. We train two BPE tokenizers, one on a uniform subset of the CLLM corpus (V_1) and one on a curated secret corpus (V_2), and select the top-k tokens that appear disproportionately often in V_1 but are rare or absent in V_2.

Stage 2: eliminating gibberish tokens. Given the identified set, we propose two approaches. (i) Tokenizer mapping deletion removes the "token $\rightarrow$ id" mappings of gibberish tokens while leaving the model unchanged, preserving all parameters including the embedding matrices. (ii) Vocabulary reduction applies cross-tokenizer distillation (Minixhofer et al.; Han et al., 2025; Minixhofer et al., 2024) to obtain a distilled LLM whose vocabulary excludes gibberish tokens; this requires weight updates but shrinks the embedding matrix and the overall parameter count. Mapping deletion is preferable when query volume is low; vocabulary reduction pays an upfront distillation cost but lowers per-request serving cost at scale.

Implication on stakeholders of secret leakage. For *online service providers* who provide secrets to users, they should be aware of the risk induced by tokenizers. For developers of code LLMs, they should proactively adopt the above mitigation strategy to mitigate the risk. For *academic researchers*, we advocate for red teaming strategies to understand every aspect of secret leakage. Such open questions include: how can an adversary *proactively* and *effectively* exploit CLLMs to extract secrets? We leave these for future explorations.

8.2 Implications on Tokenizer Design

Our study surfaces two structural drawbacks of standard BPE tokenizers:

1. **Limited flexibility.** BPE vocabulary is fixed prior to LLM pre-training; once fixed, Code LLM trainers should strictly follow the vocabulary to segment words into subwords, and adding or removing vocabulary items without full model retraining is challenging and under-explored.

2. **Sub-optimal compression utility under train-test distribution shift.** BPE has its historical origins in text compression, and it was brought to language models in 2016, since the community believes that compression boosts the performance of LLMs (Huang et al., 2024b). On LLMs, BPE is a heuristic-based algorithm to compress on the *training* distribution. Therefore, its compression rate on specific *test* distributions may degrade. In fact, we expect such a train-test distribution shift to exist for every downstream task of (Code) LLMs. Therefore, degradation of compression may affect downstream performance and the robustness of CLLMs. Our study presents a corner case for these distribution shifts, and shows the *weird* behavior of BPE.

In summary, BPE harms downstream task performance, and such defects may not be easily mitigated due to limited flexibility in vocabulary. To effectively adapt CLLMs to different downstream tasks, there have been a lot of efforts on CLLM post-training, agentic AI. We regard that an under-appreciated direction is promising: *tokenizer adaptation*.

As a final remark, most evaluations of BPE to date are empirical, and the reasons for its good practical performance are not well understood across the AI community (Kozma and Voderholzer, 2024). We then call for principled theoretical investigations towards BPE for academic researchers.

Limitations

Limited by space, this paper fails to enumerate all choices of CLLMs since there are too many. We selected a popular subset of them. Besides, among all subword-based tokenization, we focus on BPE, since it is the dominant tokenization strategy in the community. We left further investigation on Unigram and WordPiece for future research.

Acknowledgment

The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14209124) of the General Research Fund, and RGC Grant for Theme-based Research Scheme Project (RGC Ref. No. T43-513/23-N).

References

Ali Al-Kaswan and Maliheh Izadi. 2023. The (ab)use of Open Source Code to Train Large Language Models . In *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, pages 9–10, Los Alamitos, CA, USA. IEEE Computer Society.

Anthropic. 2025. Claude code: An agentic cli for developers. https://github.com/anthropics/claude-code. Official Product Announcement, Accessed: 2026-04-17.

Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 2023. SecretBench: A Dataset of Software Secrets . In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 347–351, Los Alamitos, CA, USA. IEEE Computer Society.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, and 1 others. 2021. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, pages 2633–2650.

Matteo Ciniselli, Luca Pascarella, and Gabriele Bavota. 2022. To what extent do deep learning-based code recommenders generate predictions by cloning code from the training set? In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 167–178, New York, NY, USA. Association for Computing Machinery.

Cursor. 2025. [link].

Runhan Feng, Ziyang Yan, Shiyan Peng, and Yuanyuan Zhang. 2022. Automated detection of password leakage from public github repositories. In *Proceedings of the 44th International Conference on Software Engineering*, pages 175–186.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

Gradually AI. 2026. Claude code statistics 2026: Key numbers, data & facts. https://www.gradually.ai/en/claude-code-statistics/. Accessed: 2026-04-18.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, and 1 others. 2024. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

HyoJung Han, Akiko Eriguchi, Haoran Xu, Hieu Hoang, Marine Carpuat, and Huda Khayrallah. 2025. Adapters for altering llm vocabularies: What languages benefit the most? *Preprint*, arXiv:2410.09644.

Heather Harvey. 2021. Behind github's new authentication token formats.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and zhou Xun. 2025. Over-tokenized transformer: Vocabulary is generally worth scaling. In *Forty-second International Conference on Machine Learning*.

Yizhan Huang, Yichen Li, Weibin Wu, Jianping Zhang, and Michael R Lyu. 2024a. Your code secret belongs to me: Neural code completion tools can memorize hard-coded credentials. *Proceedings of the ACM on Software Engineering*, 1(FSE):2515–2537.

Yizhan Huang, Zhe YANG, Meifang Chen, HUANG Nianchen, Jianping Zhang, and Michael R. Lyu. 2026. Data compressibility quantifies LLM memorization. *Transactions on Machine Learning Research*.

Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. 2024b. Compression represents intelligence linearly. In *First Conference on Language Modeling*.

Nvidia Hugging Face, ServiceNow. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

László Kozma and Johannes Voderholzer. 2024. Theoretical analysis of byte-pair encoding. *arXiv preprint arXiv:2411.08671*.

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.

Yinxi Li, Yuntian Deng, and Pengyu Nie. 2025. Tokdrift: When llm speaks in subwords but code speaks in grammar. *arXiv preprint arXiv:2510.14972*.

Michael Meli, Matthew R McNiece, and Bradley Reaves. 2019. How bad can it git? characterizing secret leakage in public github repositories. In *NDSS*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *Preprint*, arXiv:1301.3781.

Benjamin Minixhofer, Edoardo Maria Ponti, and Ivan Vulić. 2024. Zero-shot tokenizer transfer. *Advances in Neural Information Processing Systems*, 37:46791–46818.

Benjamin Minixhofer, Ivan Vulić, and Edoardo Ponti. Universal cross-tokenizer distillation via approximate likelihood matching. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yuqing Nie, Chong Wang, Kailong Wang, Guoai Xu, Guosheng Xu, and Haoyu Wang. 2025. Decoding secret memorization in code llms through token-level characterization. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 2880–2892. IEEE.

OpenAI. 2026. Openai codex: A series of models for code generation. `https://github.com/openai/codex`. GitHub Repository, Accessed: 2026-04-17.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. IEEE.

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*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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.

Claude Elwood Shannon. 2001. A mathematical theory of communication. In *ACM SIGMOBILE Mobile Computing and Communications Review*, volume 5, pages 3–55.

L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Authur, and et al. 2024. Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. 2024. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Weibin Wu, Haoxuan Hu, Zhaoji Fan, Yitong Qiao, Yizhan Huang, Yichen Li, Zibin Zheng, and Michael Lyu. 2025. An empirical study of code clones from commercial ai code generators. *Proceedings of the ACM on Software Engineering*, 2(FSE):2874–2896.

Yuhang Wu, Zhaoxin Zhang, Zhengyi Li, Yuan Zhang, Min Yang, Hao Zhou, Xiaofeng Wang, Linzhang Wang, Jianhua Li, Ziwen Zhu, and Xinhui Han. 5. The skeleton keys: A large-scale analysis of credential leakage in mini-apps. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society.

Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. 2024a. Unveiling memorization in code models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.

Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. 2024b. Unveiling memorization in code models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13.

Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2411–2425, New York, NY, USA. Association for Computing Machinery.

A Ethics considerations

We contacted the authors in Huang et al. (2026) to obtain the secret dataset described in Section 5. We use the secrets data solely for statistical computing (e.g., entropy). We refrain from utilizing the sensitive information at any level.

B Example calculation of entropy

Following the discussions in Section 2.2, we consider the GitHub personal access token with regular expression `ghp_[a-zA-Z0-9]{36}`. Assume the vocabulary is $\{A, \ldots, Z, a, \ldots, z, 0 \ldots 9, \_\}$. Each char from the randomized part has an equal expected number of appearance $36 \cdot \frac{1}{62} = \frac{18}{31}$.For "$g, h, p$", the expected number of appearances is $1 + \frac{18}{31} = \frac{49}{31}$. For "_", the expected number of appearances is 1. Therefore, for each of g, h, p:

$$p(g) = p(h) = p(p) = \frac{49/31}{40} = \frac{49}{1240} \approx 0.03952.$$

. For "_":

$$p(\_) = \frac{1}{40} = 0.02500.$$

. For each of the 59 "other" symbols:

$$p(\text{other}) = \frac{18/31}{40} = \frac{18}{1240} \approx 0.01452.$$

Assume the base-2 entropy, the overall entropy is

$$\begin{aligned} H &= -\sum_i p_i \log p_i \\ &= -\Big[3 \cdot \frac{49}{1240} \log \frac{49}{1240} + \frac{31}{1240} \log \frac{31}{1240} \\ &\qquad + 59 \cdot \frac{18}{1240} \log \frac{18}{1240}\Big] \\ &= 5.915 \quad \text{bits.} \end{aligned}$$

For the sample space size (i.e., vocabulary size) 63, the maximal entropy is achieved by a uniform distribution.

$$H_{\max} = -\sum_i \frac{1}{63} \log \frac{1}{63} = 5.977.$$

The normalized entropy is

$$H/H_{\max} = 5.915/5.977 = 0.9896$$

Through the example, we learn that a GitHub personal access token achieves around 99% of the maximal entropy.

C Character-wise Tokenization as a Mitigation

Gibberish bias is grounded on the discrepancy of how secret strings are processed at the design stage (char-level) and the CLLM inference stage (token-level). Therefore, a promising solution to address this gap is to force character-wise tokenization for

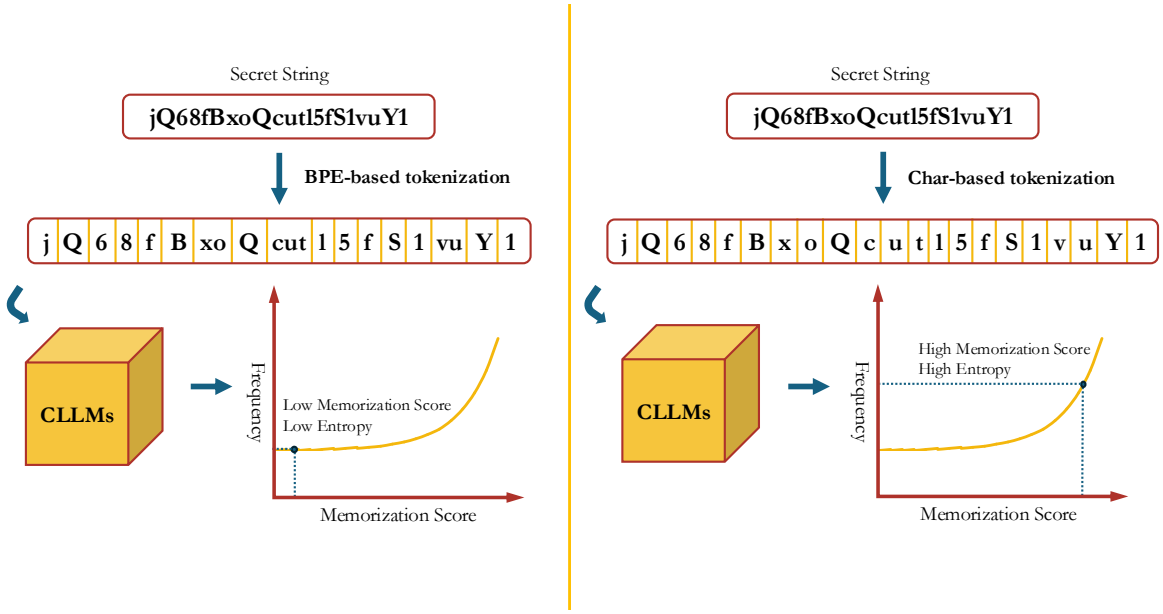

Figure 6: Mitigation Strategy Visualized

code secrets. Recent research (Vieira et al., 2025) demonstrates that token-level models can be reinterpreted at the character level through a search-based alignment algorithm. By reconstructing conditional distributions over characters and employing beam search pruning for efficiency, this approach provides a principled means to approximate character-level behavior, thereby mitigating discrepancies in model processing. Figure 6 illustrates the overall idea.

The strategy aligns the discrepancy between the two stages. Therefore, we expect it to eliminate gibberish bias. The strategy also draws inspiration from digit-wise tokenization on integers in general-purpose LLMs, adopted in Llama-1 (Touvron et al., 2023a), Llama-2 (Touvron et al., 2023b), and Mistral (Jiang et al., 2023), aiming to explore arithmetic-related capabilities of LLMs.

D Tokenization Heatmaps on Unigram-based Tokenizers

To check whether the counterintuitive tokenization behavior is specific to BPE, we repeat the 2-char enumeration experiment from Section 3.1 on two LLMs that adopt Unigram-based tokenization (Kudo and Richardson, 2018): XLNet and T5. Figure 7 shows that similar non-uniform token-splitting patterns persist, indicating that gibberish bias is not an artifact of BPE alone but a more general consequence of data-driven subword tokenization.

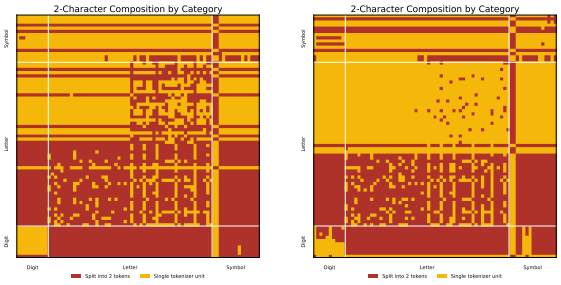

Figure 7: 2-char sub-string tokenization on unigram-based tokenizers: XLNet (left) and T5 (right). Axes and color encoding follow Figure 3.