Understanding Secret Leakage Risks in Code LLMs: A Tokenization Perspective

Meifang Chen* **Zhe Yang†** **Huang Nianchen‡** **Yizhan Huang*** **Yicen Li*** **Michael R. Lyu***

Abstract

Code secrets are sensitive assets for software developers, while their leakage imposes high risks to cybersecurity. While the rapid development of AI code assistants powered by Code Large Language Models (CLLMs) revolutionizes the landscape of software engineering, CLLMs are shown to inadvertently leak such secrets due to a notorious memorization phenomenon. This study firstly reveals that Byte-Pair Encoding (BPE) tokenization leads to unexpected behaviour of secret memorization. Specifically, we discover that some secrets are among the easiest for CLLMs to memorize, even though they look random. Our investigation reveals that these secrets exhibit high character-level entropy, but low token-level entropy. We coin the phenomena as *gibberish bias*. We identified the root of the bias as 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 for the current tokenizer design.

1 Introduction

Code Large Language Models (CLLMs) are revolutionizing the entire software development lifecycle, impacting every phase from requirements analysis to deployment and maintenance. Recent academic advances, such as Qwen2.5-Coder [20], StarCoder2 [19], and Deepseek Coder [12], demonstrate significant improvements in code generation, understanding, and reasoning across multiple programming languages and tasks. These models have been evaluated on rigorous benchmarks like HumanEval [4], LiveCodeBench [21], and others, showing state-of-the-art performance in code synthesis, completion, and bug repairs. On the commercial side, tools like GitHub Copilot [10], Amazon CodeWhisperer [2], and Cursor [6] 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 are widely adopted, for instance, GitHub Copilot boasts over 15 million users, demonstrating a substantial increase of more than 4x year-over-year [7]. It is reshaping developer workflows, reducing the need for boilerplate coding, and enabling higher-level abstraction in software design.

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 [42, 40, 29]. 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

*The Chinese University of Hong Kong
†Nanyang Technological University
‡University of Southern California

The Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Deep Learning For Code in the Agentic Era.

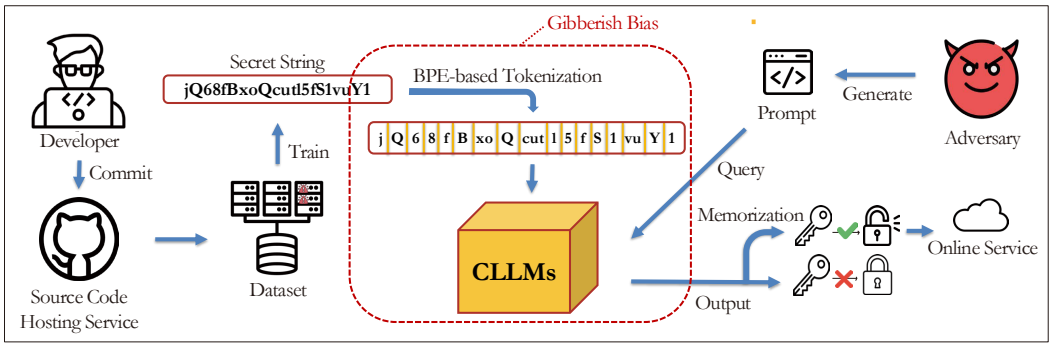

Figure 1: The risk map of secret leakage through CLLMs. The red box highlights the findings in this papers.

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 pivot 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 [17]. 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 token sequences, significantly reducing the difficulty. 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 border implications for tokenizer design.

Contribution summary.

1. This paper identifies a new risk in secret leakage through CLLMs: 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 evaluates that the secret leakage risk is expected to manifest more under current "larger tokenizer" trend.
4. The paper discusses the potential mitigation strategies, and broader implications to the community.

2 Background

2.1 Tokenization of Code LLMs

In NLP, early explorations use word-level tokenization (e.g., word2vec [25]), 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 [25]. Later, researchers developed subword tokenization strategies that segment words into smaller units. The process typically include 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, while strictly adhering to the pre-tokenization rules.

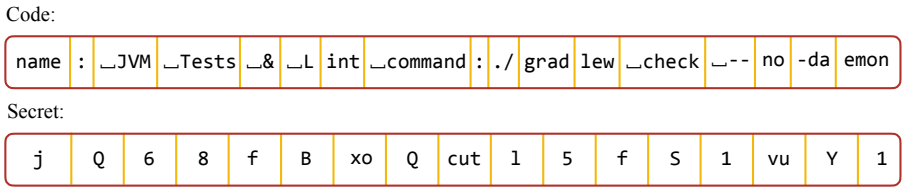

Figure 2: Use BPE to tokenize one line of normal code and secrets. Each token is a subword.

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 followed during all stages of LLM training (i.e., pre-training, post-training). During CLLM training and inference, only stages (1) and (3) are applied. In decoder-only LLMs, a tokenizer assigns an input string into a sequence of numerical IDs, which are further transformed into embeddings via an embedding matrix.

The core of tokenization is the vocabulary construction algorithm, and the Byte-Pair Encoding (BPE) [31, 9] is the de facto standard for LLMs since GPT-2 [30]. 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, StarCoder2, Qwen2.5-Coder, generally employs BPE-based tokenization, following the tokenizers on LLMs. For CLLM tokenizers, there are slight changes towards code-related tasks. For example, many Code LLMs adapt vocabularies on code-base to improve generation fidelity and ensure compilable output [38]. 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 them to online code hosting services like GitHub. Recent studies have shown that a vast amount of secrets are exposed in public software repositories [3, 8, 24, 44, 41]. 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}`
GitHub Personal Access Token	`ghp_[0-9a-zA-Z]{36}`
Facebook Access Token	`EAACEdEose0cBA[0-9A-Za-z]+`

An important property of code secrets is that secrets typically exhibit high (char-level) entropy [32]. For human being, the strings look like gibberish. High entropy indicates high uniqueness. Consequently, for online service providers, entropy is a pivotal design consideration on the security of their secrets. Secrets typically follows a specifc 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 either be a char or 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

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

Taking GitHub personal access tokens [iv] [14] 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 maximal entropy 5.977 bits. We defer the detailed calculation to appendix B.

3 Motivating Study: BPE Is *Insane* 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 token-level. This section then presents a motivating study using visualizations, and reveals that BPE indeed results in *insane* behaviors.

3.1 Case Study

This case study discusses the tokenization example in Figure 2. The figure illustrates how Deepseek Coder [12] tokenizes a part of normal code and a secret (substring). We observe significant difference on **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 in length 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

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.

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 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 presents 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),

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

(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.

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 math notations. Inspired by several research works on secret memorization, we characterize gibberish bias by memorization. Secret leakage has strong implications to the community, since upon leakage, these credentials may grant an adversary access to online services, bringing severe cybersecurity concerns.

4.1 Preliminaries: Entropy-Memorization Law

A recent LLM memorization work [17] has shown that entropy, the essential property that comes with the design of code secrets, is closely related to memorization. The work discovers 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 on the outcome space $\mathcal{V}$. Since LLMs process strings at token-level, $\mathcal{V}$ is defined over tokens. Moreover, in Entropy-Memorization Law, the authors adopted a level-set based calculation for $\mathcal{V}$, where we skip due to space limit.

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 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 Experiments

Experimental Setup. We adopt the same experimental setup with [17]. We use OLMo-1B [11], a fully-open LLM with its training corpus Dolma [33]. In the experiments, we sampled 240k sequences from Dolma. We reproduce the results with the same algorithm adopted in [17]. 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 ESVXlO3url9YWlFW/TA9cshCEhtu7IKJ/p5soJ/gGpj7vbvFrAY/eIioQ6Dw23KjZ
2 PszYzqs83S3E5mpi7/y/8M9eC90MRTZfduQOYW76ig6SOSPNe41IG5LoP3FGBn2N0Rj
3 5laXo6c1IbEbegDmzGPwGNTsHZmEy6QGznu5Sh2UCWvueywb2ee+CCE4zQiZstxU9
```

Listing 1: Three examples of gliberish generated by OLMo-1B.

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

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

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 has significantly lower entropy (8.804) 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 on 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 he 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.

The above results reveals an essential defect induced by tokenization. The findings leave an intriguing and important question: How does this happen? To explore the question, we discusses the design issues on BPE, the tokenization strategy on every state-of-the-art (Code) LLMs in the next section.

5 How Does Tokenization Induce Gibberish Bias?

Experimental Setup The following experiments include tokenizers of three representative Code LLMs: Deepseek Coder [12], Qwen2.5-Coder [20], and StarCoder2 [19]. The experiments involves comparisons on two datasets: the "secret" dataset as introduced in Section 4 and the subsample Stack V2 dataset [19]. Stack V2 is the training corpus of StarCoder2.

[v] 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.

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?*

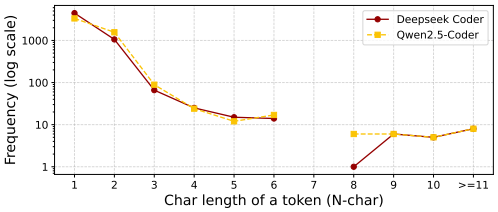

Figure 4: The distribution of n-char tokens.

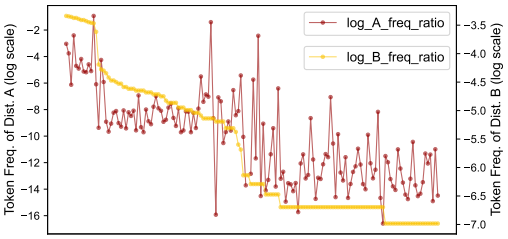

Figure 5: Token distributions on the subsampled Stack V2 dataset (Distribution A, in brown) and the secret dataset (Distribution B, in yellow).

Following the idea 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 [32], 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 secret dataset by frequency, and compares 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 long-tail distribution explains the low entropy of secrets, hence explains the claimed gibberish bias. Further investigation confirms the significant distribution shift between secret data and general CLLM training data.

6 Gibberish Bias With a Larger Tokenizer Vocabulary

There is a growing consensus in both academia and industry that larger models shall benefit from a larger vocabulary [34, 15]. As the prevailing trends of building larger Code LLMs, it is believed that CLLMs deserve large vocabularies. Tao et al.'s [34] 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* [34] 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 as *SC-3B-o*, *SC-7B-o*, and *SC-15B-o*, correspondingly.

With three new tokenizers, we investigate gibberish bias on the secret dataset. Following [17], we report entropy and normalized-entropy of these secrets introduced in Section 2.2 at the token-level.

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

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 k eeps deviating from the baseline ratio 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

With the widespread adoption of AI-powered coding assistants, the memorization of sensitive data by CLLMs is a growing concern. Initial research focused on empirically demonstrating and quantifying this risk. HCR [16] 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. [43] provided a systematic study showing CLLMs memorization a broad scope of data related to code. A study [39] investigates how memorization leads to general code cloning, raising concerns about copyright infringement and bug propagation. Other studies explore code clone as implcations of memorization [5, 1]. Recent research probes the underlying mechanisms at a finer granularity. For example, DESEC [28] addresses the problem from the decoding stage. However, these prior works do not address the root cause of why some secrets are so readily memorized. Our work addresses this fundamental issue from the perspective of the tokenization process. We explore 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 the Security of Secrets

Mitigating gibberish bias. Gibberish bias is grounded on the discrepancy of how secret strings are processed at design stage (char-level) and CLLM inference stage (token-level). Therefore, a

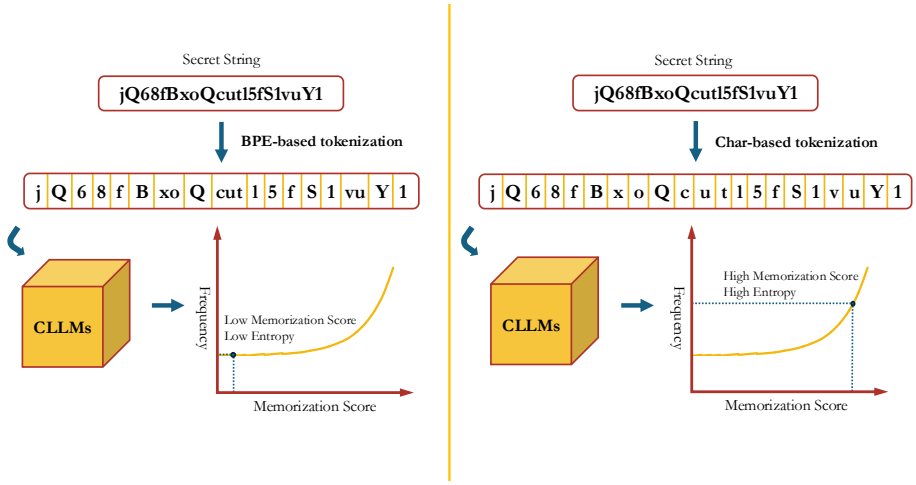

Figure 6: Mitigation Strategy Visualized

promising solution to fix the gap and mitigate this to force character-wise tokenization for code secrets. Recent research [37] 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. Fig 6 illustrates the overall idea.

The strategy aligns the discrepancy between two stages. Therefore, we expect it to eliminate the gibberish bias. The strategy also draws inspiration from digit-wise tokenization on integers in general-purpose LLMs. Such strategy is adopted in some LLMs Llama-1 [35], Llama-2 [36], and Mistral [22], aiming to arithmetic-related capabilities of LLM. Figure 6 presents the visualized mitigation strategy.

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 aspects 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 re-training 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 [18]. 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 hurts 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*. These methods enable a change of vocabulary without full re-training. Representative methods include [13, 26, 27].

As our final remark, most evaluations of BPE to date are empirical, and the reasons for its good practical performance are not well understood in the whole AI community [23]. We then call for principled theoretical investigations towards BPE for academic researchers.

References

[1] Ali Al-Kaswan and Maliheh Izadi. 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, May 2023. IEEE Computer Society. doi: 10.1109/NLBSE59153.2023.00008. URL https://doi.ieeecomputersociety.org/10.1109/NLBSE59153.2023.00008.

[2] Amazon. AI code generator: Amazon Code Whisperer, July 2023. URL https://aws.amazon.com/codewhisperer/.

[3] Setu Kumar Basak, Lorenzo Neil, Bradley Reaves, and Laurie Williams. 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, May 2023. IEEE Computer Society. doi: 10.1109/MSR59073.2023.00053. URL https://doi.ieeecomputersociety.org/10.1109/MSR59073.2023.00053.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[5] Matteo Ciniselli, Luca Pascarella, and Gabriele Bavota. 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, 2022. Association for Computing Machinery. ISBN 9781450393034. doi: 10.1145/3524842.3528440. URL https://doi.org/10.1145/3524842.3528440.

[6] Cursor, Jul 2025. URL https://docs.cursor.com/welcome.

[7] Entrepreneur. Github surpasses over 15 million users: Microsoft, May 2025. URL https://www.entrepreneur.com/en-in/news-and-trends/github-surpasses-over-15-million-users-mi

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

[9] Philip Gage. A new algorithm for data compression. *C Users J.*, 12(2):23–38, February 1994. ISSN 0898-9788.

[10] GitHub. Github Copilot: Your AI pair programmer, July 2023. URL https://github.com/features/copilot.

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

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

[13] HyoJung Han, Akiko Eriguchi, Haoran Xu, Hieu Hoang, Marine Carpuat, and Huda Khayrallah. Adapters for altering llm vocabularies: What languages benefit the most?, 2025. URL https://arxiv.org/abs/2410.09644.

[14] Heather Harvey. Behind github's new authentication token formats, Apr 2021. URL https://github.blog/2021-04-05-behind-githubs-new-authentication-token-formats/.

[15] Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and zhou Xun. Over-tokenized transformer: Vocabulary is generally worth scaling. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=gbeZKej40m.

[16] Yizhan Huang, Yichen Li, Weibin Wu, Jianping Zhang, and Michael R Lyu. 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, 2024.

[17] Yizhan Huang, Zhe Yang, Meifang Chen, Jianping Zhang, and Michael R. Lyu. Entropy-memorization law: Evaluating memorization difficulty of data in llms. *arXiv preprint arXiv:2507.06056*, 2025.

[18] Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. Compression represents intelligence linearly. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=SHMj84U5SH.

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

[20] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[21] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

[22] 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. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

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

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

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

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

[27] Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. Universal Cross-Tokenizer Distillation via Approximate Likelihood Matching, May 2025. URL http://arxiv.org/abs/2503.20083. arXiv:2503.20083 [cs].

[28] Yuqing Nie, Chong Wang, Kailong Wang, Guoai Xu, Guosheng Xu, and Haoyu Wang. Decoding secret memorization in code llms through token-level characterization. *arXiv preprint arXiv:2410.08858*, 2024.

[29] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 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, 2022.

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

[31] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/.

[32] Claude Elwood Shannon. A mathematical theory of communication. In *ACM SIGMOBILE Mobile Computing and Communications Review*, volume 5, pages 3–55, 2001. doi: 10.1145/584091.584093.

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

[34] Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=sKCKPr8cRL.

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

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

[37] Tim Vieira, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell. From language models over tokens to language models over characters. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=sQS0roNQZR.

[38] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.685. URL https://aclanthology.org/2021.emnlp-main.685/.

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

[40] Weibin Wu, Haoxuan Hu, Zhaoji Fan, Yitong Qiao, Yizhan Huang, Yichen Li, Zibin Zheng, and Michael Lyu. An empirical study of code clones from commercial ai code generators. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3729397. URL https://doi.org/10.1145/3729397.

[41] Yuhang Wu, Zhaoxin Zhang, Zhengyi Li, Yuan Zhang, Min Yang, Hao Zhou, Xiaofeng Wang, Linzhang Wang, Jianhua Li, Ziwen Zhu, and Xinhui Han. 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, February 5. URL https://www.ndss-symposium.org/ndss-paper/the-skeleton-keys-a-large-scale-analysis-of-creden

[42] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. Unveiling memorization in code models. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639074. URL https://doi.org/10.1145/3597503.3639074.

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

[44] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 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, 2023. Association for Computing Machinery. ISBN 9798400700507. doi: 10.1145/3576915.3616591. URL `https://doi.org/10.1145/3576915.3616591`.

A Ethics considerations

We contacted the authors in [17] to obtain the sampled dataset described in 4. We use the secrets data solely for statistical computing (e.g. entropy). We retrain 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 equal expected number of appear $36 \cdot \frac{1}{62} = \frac{18}{31}$.For "$g, h, p$", the expected number of appearance is $1 + \frac{18}{31} = \frac{49}{31}$. For "_", the expected number of appearance 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

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

For the sample space size (i.e., vocabulary size) 63, the maximal entropy is achieved by 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 GitHub personal access token achieve around 99% of the maximal entropy.