Uncovering Pretraining Code in LLMs: A Syntax-Aware Attribution Approach

Yuanheng Li[1*], Zhuoyang Chen[1*], Xiaoyun Liu[1], Yuhao Wang[1], Mingwei Liu[2], Yang Shi[1], Kaifeng Huang[1†], Shengjie Zhao[1]

[1]School of Computer Science and Technology, Tongji University
[2]School of Software Engineering, Sun Yat-sen University
[1]{2253551, 2253721, 2354269, 2251052, shiyang, kaifengh, shengjiezhao}@tongji.edu.cn
[2]liumw26@mail.sysu.edu.cn

Abstract

As large language models (LLMs) become increasingly capable, concerns over the unauthorized use of copyrighted and licensed content in their training data have grown, especially in the context of code. Open-source code, often protected by open-source licenses (e.g., GPL), poses legal and ethical challenges when used in pretraining. Detecting whether specific code samples were included in LLM training data is thus critical for transparency, accountability, and copyright compliance. We propose SYNPRUNE, a syntax-pruned membership inference attack method tailored for code. Unlike prior MIA approaches that treat code as plain text, SYNPRUNE leverages the structured and rule-governed nature of programming languages. Specifically, it identifies and excludes consequent tokens that are syntactically required and not reflective of authorship from attribution when computing membership scores. Experimental results show that SYNPRUNE consistently outperforms the state-of-the-arts.Our method is also robust across varying function lengths and syntax categories.

Code and Dataset —
https://anonymous.4open.science/r/SYNPRUNE-FED7

Introduction

The fundamentals of large language models (LLMs) stems from the vast pre-training datasets. However, the copyright issue in the training corpora deserves serious attention. Recently, several legal disputes have arisen concerning the use of copyrighted materials in training data (Grynbaum and Mac 2025; Vynck 2025). Respecting copyright and intellectual property is essential, not only as a legal obligation enforced by governments, but also as a means of fostering supportive and respectful environments for content creators. As LLMs proliferate and AI-generated content grows, safeguarding human creativity is vital to preserving unique and diverse perspectives.

LLMs' coding abilities have enabled applications from autonomous agents to automated coding. However, these coding abilities stem from large-scale training on open-source code, which often comes with specific usage obligations. However, such code is typically governed by specific usage obligations. A well-known example is the copyleft license (*e.g.,* GPL (Foundation 2025a)). The GPL was introduced in 1989 as part of the GNU Project (Foundation 2025a). Eighteen years later, we witnessed the first recognized lawsuit for a GPL violation (lvcriminaldefense 2024; Informationweek 2024), when Monsoon Multimedia was sued by the Software Freedom Conservancy. The following decades have seen many GPL conflicts and litigations (wikipedia 2024), which underscores the importance of respecting copyrighted content (Xu et al. 2024). We anticipate that similar open-source litigation may soon target LLM vendors. In fact, disputes over the use of code for pretraining have already emerged (LLP 2025), and few have reached resolution due to the protracted nature of litigation. As a result, the transparency and accountability questions in LLMs remain unanswered.

Several studies attempt to answer the transparency and accountability issues in LLMs. Zhou et al. (Zhou et al. 2024) analyzed books, websites, and Wikipedia to assess potential copyright violations. Recent benchmarks (Chen et al. 2024; Liu et al. 2024) support evaluation of both literal copying (*e.g.,* verbatim reproduction) and non-literal copying (*e.g.,* replicated plots or characters) from LLM outputs. There are also proactive measures known as copyright traps (Shilov, Meeus, and de Montjoye 2024; Meeus et al. 2024), which embed crafted copyrighted content to enable precise copyright violation detection. However, this approach is limited by the use of pre-inserted traps rather than those derived from real-world scenarios.

Another approach involves applying membership inference attacks (MIAs) to LLMs, which initially determine whether a specific data point was part of the training dataset of a machine learning model (Shokri et al. 2017; Song and Mittal 2021; Song, Shokri, and Mittal 2019; Huang, Gong, and Reiter 2024; Yeom et al. 2018; Leino and Fredrikson 2020; Long et al. 2018; Nasr, Shokri, and Houmansadr 2019; Sablayrolles et al. 2019; Salem et al. 2018; Truex et al. 2019), thereby exposing potential privacy risks. MIAs in LLMs share the same intuitions of MIAs in machine learning that models memorize data (overfitting) (Song and Shmatikov 2019; Carlini et al. 2021; Tirumala et al. 2022). Tirumala et al. (Tirumala et al. 2022) find that models memorize nouns and numbers,

[*]Both authors contributed equally to this work.
[†]Kaifeng Huang is the corresponding author.

and that larger models can memorize a larger portion of the data before over-fitting. These methods assume access to pretraining data to train shadow or reference models. Yang et al. (Yang et al. 2024) presented GOTCHA, a membership inference attack using surrogate models tailored for code LLMs. However, training such models is computationally expensive, and pretraining data is often unavailable, especially for commercial LLMs. Differently, several MIAs on LLMs typically exploit the model's token-level outputs and probabilities (Carlini et al. 2021), often in combination with techniques such as synonym substitution to leverage neighboring data (Mattern et al. 2023), detection of low-probability tokens (Shi et al. 2023), or calibration of token probabilities using term frequency divergence (Zhang et al. 2024). However, their effectiveness is still limited, as Duan (Duan et al. 2024) found that MIAs on LLMs perform only marginally better than random guessing. In addition, most existing MIAs for LLMs target general natural language text rather than code.

Although existing text-based MIA techniques can be applied to code, their full potential remains untapped. A key distinction between code and natural language is the presence of formal syntax and structured constructs. When developers write code, they reason about task logic while adhering to the programming language's syntax rules. We hypothesize that *the effectiveness of MIA techniques on code-related LLMs is limited by their failure to leverage these syntactic characteristics*. Through our observations, we find that certain syntax elements (*e.g.,* Data models, expressions, and statements) follow syntax conventions, rather than expressing individual authorship. That is, once specific syntax *conditions* are met, certain *consequent tokens* must or are highly likely to appear. Therefore, *consequent tokens* should be excluded from attribution when computing MIA scores. Based on this insight, we propose SYNPRUNE, a syntax-pruned membership inference attack (MIA) method tailored for code. We summarize a set of 47 syntax conventions derived from the official Python language reference.

We evaluate SYNPRUNE on a curated benchmark of Python functions that are authentic and verified, as existing benchmark on code only provides synthetic or assumed members and non-members. In our benchmark, Member functions are sourced from the Pile dataset, which is widely used in the pretraining of many LLMs. Non-member functions are collected from sources published after the release cut-off dates of the evaluated LLMs, ensuring that none of the models have encountered them during training. Experimental results show that SYNPRUNE outperforms state-of-the-art MIA techniques, achieving an average AUROC improvement of 15.4% across four models and three member-non-member ratios. Additionally, we demonstrate that SYNPRUNE remains robust across varying function lengths. Through ablation studies, we assess the contribution of each syntax convention category.

Contribution. We make the following contributions in detecting pretraining code in LLMs.

- We propose SYNPRUNE, a syntax-pruned membership inference attack method for code, which excludes from attribution tokens that are inherently determined by Python syntax conventions.
- We present a benchmark comprising authentic and verifiable member and non-member functions, offering a reliable ground truth often absent in existing approaches.
- We evaluate the effectiveness of SYNPRUNE against state-of-the-art approaches, examine its robustness across varying function lengths, and analyze the impact of different syntax convention categories through ablation studies.

Related Work

MIA in Machine Learning. Membership inference attacks (MIAs) have gained traction as tools to audit privacy risks in machine learning (Song and Mittal 2021; Song, Shokri, and Mittal 2019; Huang, Gong, and Reiter 2024; Yeom et al. 2018; Leino and Fredrikson 2020; Long et al. 2018; Nasr, Shokri, and Houmansadr 2019; Sablayrolles et al. 2019; Salem et al. 2018; Truex et al. 2019). Shokri et al. (Shokri et al. 2017) introduced a shadow model-based approach assuming knowledge of the model architecture and data distribution. Subsequent works relaxed or challenged these assumptions. Salem et al. (Salem et al. 2018) used diverse shadow model architectures; Liu et al. (Liu et al. 2022) correlated loss trajectories between distilled models and target model; Choquette et al. (Choquette-Choo et al. 2021) assumed confidence scores are unavailable. Rezaei et al. (Rezaei and Liu 2021) highlighted evaluation pitfalls between overfitted and well-trained models. Kazmi et al. (Kazmi et al. 2024) assume synthetic non-member data to be consistent with queried non-member data. Ye et al. (Ye et al. 2022) formalized attack assumptions in a unified framework. Besides, the shadow model framework has seen other advances. Yeom et al. (Yeom et al. 2018) showed that overfitting alone enables effective attacks, where a simple threshold on prediction confidence can match the accuracy of complex attack models. Song et al. song2021systematic highlighted the lack of per-sample analysis, prompting difficulty calibration methods that adjust membership scores based on sample hardness (Watson et al. 2021; He et al. 2024). Carlini et al. (Carlini et al. 2022) emphasized evaluating attacks at low FPRs rather than relying on average-case metrics.

MIA in Language Models. Song et al. (Song and Shmatikov 2019) investigated how deep learning-based text generation models memorize training data, while Carlini et al. (Carlini et al. 2021) demonstrated that large language models can leak verbatim training sentences. While literal (verbatim) memorization is easier to detect, identifying non-literal memorization remains challenging. Most approaches focus on modeling the likelihood of word sequences in context. Duan (Duan et al. 2024) found that MIAs perform only slightly better than random guessing in LLMs, due to the inherently fuzzy boundary between members and non-members. To address this, Mattern et al. (Mattern et al. 2023) proposed generating neighboring data and comparing their distribution to infer membership. Zhang et al. (Zhang et al. 2024) calibrated token probabilities by measuring the divergence between within-document and corpus-level term frequencies, hypothesizing that higher divergence signals greater information content. Shi et al. (Shi et al. 2023) introduced a reference-free method, MIN-K% prob, which

Category	Syntax Nodes	Example Syntax Node	Conditional Token	Consequent Token
Data Model	List, Slice, Dict, Set, String, Bytes, Object, Tuple	List	'['	']'
		Dict	'{'	'}'
		Tuple	'('	')'
Expression	call, lambda, conditional, comprehension, Chained Comparison	call	*identifier*(	')'
			identifier(*identifier*	','
		conditional	*expr* `if` *cond*	`else`
		chained comparison	*expr comp_op expr*	*comp_op*
Single Stmts	`import`, `assert`, `global`	`import`	`import` *module*	`as`
			`from` *module*	`import`
Compound Stmts	`if`, `for`, `try`, `with`, `class`, function, `while`, `match`	`for`	`for` *target_list*	`in`
			`for` *target_list* `in` *starred_list*	':', `[SP]`, `[IND]`, `[BR]`
		`if`	`if` *assignment_expression*	':'
		`try`	`try:` *suite* `except` *expression*	`as`
		function	`def` *funcname*(*identifier*	',', ')'
		`with`	`with` *with_item*	`as`
		`while`	`while` *condition*	':'
		`match`	`match` *subject*	':'

Table 1: List of the Collected Syntax Conventions (`[SP]`, `BR`, and `[IND]` denote the space, line break, and indentation, resp.)

assumes members are less likely to contain low-probability tokens compared to non-members.

Copyright Sources in Language Models. Xu et al. (Xu et al. 2024) found that LLMs fail to respect copyright embedded in user inputs. Zhou et al. (Zhou et al. 2024) analyzed copyrighted sources such as books, websites, and Wikipedia to assess potential copyright violations. Recent benchmarks (Chen et al. 2024; Liu et al. 2024) support evaluation of both literal copying (*e.g.,* verbatim reproduction) and non-literal copying (*e.g.,* replicated plots) in LLM outputs. Document memorization in LLMs is another challenge. Meeus et al. (Meeus et al. 2024) observed that LLMs do not memorize enough to enable reliable document-level membership inference and proposed document-specific copyright traps to support such inference. Shilov et al. (Shilov, Meeus, and de Montjoye 2024) introduced fuzzy copyright traps designed to evade deduplication and persist through training. Interestingly, despite the well-established nature of code copyright—supported by numerous litigations and enforcement cases, there has been relatively little research on copyright issues in code-generating LLMs. Yang et al. (Yang et al. 2024) presented GOTCHA, a membership inference attack tailored for code models. Similarly, Wan et al. (Wan et al. 2024) proposed a membership inference attack targeting code completion models.

Methodology

Preliminary

Source code is considered a "literary work" under copyright law. The "literary work" can be viewed as a sequence of tokens arranged to reflect the author's intentions. Existing MIA techniques (Meeus et al. 2024; Yang et al. 2024; Wan et al. 2024) treat all source code tokens equally and compute membership probabilities of an LLM based on a sample of these tokens. Generally, it is recognized that elements such as variable names, data and control flows, and API usage, together with the involved variables, embody the author's intent and serve as indicators of source code authorship. However, we observe that some tokens primarily serve a grammatical purpose, functioning to complete the syntax constraint of the programming language.

Syntax Conventions

We chose Python as our target language for studying syntax conventions, as it is a widely used programming language with broad applications in data analysis, artificial intelligence, and scientific computing. Syntax conventions refer to the specific rules defined by the Python language that programmers *must* follow. Failure to do so will result in syntax errors and prevent the code from executing. Therefore, we consider tokens that follow syntax conventions to be highly predictable patterns and exclude them from the MIA calculation.

In the example provided in our overview, the programmer completes the task by declaring a method named `add` within an existing class, following both the task specification and the language's syntax requirements. The method contains various syntax elements, such as a method declaration and a *for* statement. The token `self` represents the class instance and is mandatory. Likewise, tokens such as the closing parenthesis `)` and colon `:` after the `row` parameter, along with the `[BR]` and `[IND]`, are also required by Python's grammar and must be present.

To compile a comprehensive collection of syntax conventions, two of our authors manually reviewed the official Python documentation (Foundation 2025b). Specifically, we referred to the Python Language Reference section of the

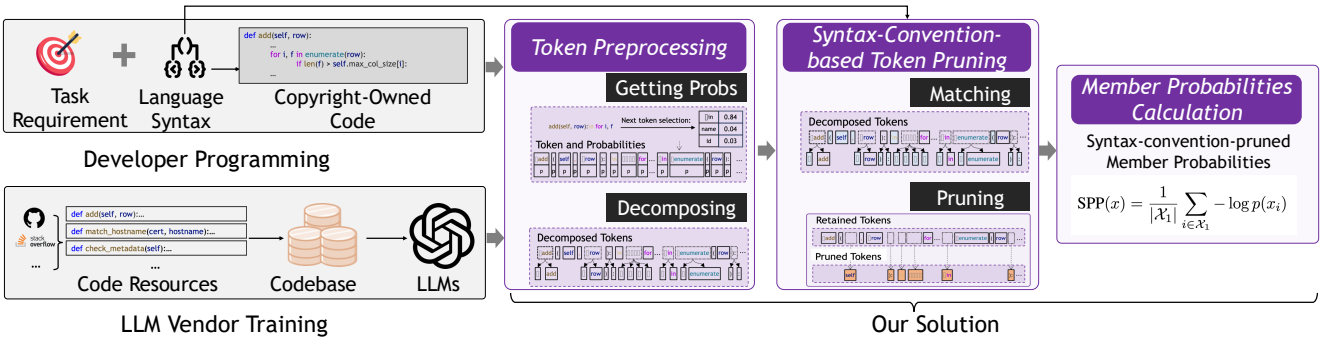


Figure 1: Illustration of our Approach Motivation and Proposed Solution

Python 3.11 documentation, released in October 2022. The authors reviewed sections of the documentation that define syntax elements, including the data model (Section 3), the import system (Section 5), expressions (Section 6), as well as statements, and function and class definitions (Sections 7 and 8). A third author is involved to resolve any arguments or disagreements.

Table 1 lists our summarized syntax conventions. We categorize them into four categories according to their syntax element types. i.e., data model, expressions, single statements and compound statements. The syntax convention is defined as a tuple $\langle \text{condition}, \text{consequence} \rangle$, where condition represents a prefix under which certain code elements conform to specific language syntax, and consequence denotes a suffix that must or high-likely to occur given the condition. In total, we obtained 47 syntax conventions, denoted as $\mathcal{C}$.

- *Data Model:* Must terminate with appropriate delimiters or accessors for syntactic completeness (*e.g.,* closing brackets ‘]’ or dot notation)
- *Expressions:* Must follow function identifiers and include a parenthesized, comma-separated list of arguments (*e.g.,* ‘)’).
- *Simple Statement*: Probably with aliases when including external packages (*e.g.,* `import` *module* `as`)
- *Compound Statement:* These statements must be followed by a colon (‘:’) and an indented block. The enclosed statements start with an indentation. The compound statements are probably extended with additional branches such as `elif`, `else`, `except`, `except*`, or `finally`. The `with` statement is typically used with aliases when wrapping code defined by a context manager. Classes and methods are defined using parameters enclosed in parentheses, followed by a colon. Method definitions begin with a `self` parameter to refer to the instance.

Syntax-Pruned MIA

We propose SYNPRUNE, a syntax-pruned membership inference attack method for inferencing code members. We hypothesize that while programmers express creativity in writing code, certain tokens inevitably occur due to syntax conventions, rather than as a reflection of individual authorship. Our approach can be divided into three phases.

The first phase is *token preprocessing*. We begin by feeding the target querying code (*e.g.,* a function) into the evaluated LLM. Using the model’s embedded tokenizer, we obtain the tokenized sequence along with the predicted probability (derived from the *logits*) for each token. The resulting tokens are denoted as $\mathcal{X}$, where each token $x_i \in \mathcal{X}$ represents the token at position i. These tokens are split according to the tokenizer’s internal logic, such as byte pair encoding (BPE) (Shibata et al. 1999) or similar subword segmentation algorithms (Li et al. 2021). However, to align with our predefined syntax conventions, we further decompose certain tokens. If a token contains sub-tokens that match any elements defined as condition or consequence in the constraint set $\mathcal{C}$, we split it accordingly. Concretely, let $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ be the original tokens, $\mathcal{X}' = \{x_{ij}\}$ be the decomposed sub-tokens. We define the split function:

$$\text{Split} : \mathcal{X} \rightharpoonup \text{List}(\mathcal{X}') \tag{1}$$

where each $x_i \in \text{dom}(\text{Split})$ is mapped to an ordered list of sub-tokens $[x_{i1}, x_{i2}, ..., x_{im}]$. dom is the domain function that includes the set of inputs for function. Each original token is equal to the string concatenation of its sub-tokens:

$$x_i = x_{i1} \oplus x_{i2} \oplus ... \oplus x_{im} \tag{2}$$

The complete set of sub-tokens is

$$\mathcal{X}' = \bigcup_{x_i \in \text{dom}(\text{Split})} \text{Split}(x_i) \tag{3}$$

The second phase is *syntax-convention-based token pruning*. Given the decomposed sub-tokens $\mathcal{X}'$, we examine each sub-token x_{ij} for tokens $x_i \in \text{dom}(\text{Split})$, and check whether it matches either c.condition or c.consequence for each constraint $c \in \mathcal{C}$. The matching process first aligns a candidate token with c.consequence and records its token offset (*i.e.,*, `lineno` for the starting character position and `col_offset` for the token length). Once a match is found, SYNPRUNE traverses the Abstract Syntax Tree (AST) parsed using Python’s `ast` module to locate the corresponding AST node based on the recorded character position and token length. Finally, SYNPRUNE validates the associated conditional tokens according to the AST hierarchy.

If all sub-tokens $[x_{i1}, x_{i2}, \ldots, x_{im_i}]$ match elements in $\mathcal{C}$, we label the original token x_i as 0 (i.e., pruned), otherwise

as 1 (i.e., retained). The function ℓ assigns a binary label to each token $x_i \in \mathcal{X}$, where $\ell(x_i) \in \{0, 1\}$.

$$\ell : \mathcal{X} \to \{0, 1\} \tag{4}$$

The third phase is *member probabilities calculation*. The Syntax-Pruned MIA Probabilities (SPP) is defined as:

$$\text{SPP}(x) = \frac{1}{|\mathcal{X}_1|} \sum_{i \in \mathcal{X}_1} -\log p(x_i) \tag{5}$$

where $\mathcal{X}_1 = \{x_i \in \mathcal{X} \mid \ell(x_i) = 1\}$. We set a threshold ϵ to determine code membership. If $\text{SPP}(x) > \epsilon$, the sample is predicted as a member. Otherwise, it is classified as a non-member.

Benchmark Construction

Existing MIA benchmarks, such as WikiMIA (Shi et al. 2023) and MIMIR (Duan et al. 2024), are designed to evaluate membership inference attacks on general textual data, rather than source code. The source code benchmark used in Yang et al.(Yang et al. 2024) consists of members and non-members randomly sampled from CodeXGLUE(Lu et al. 2021), with labels assigned afterward. We argue that such a design fails to reflect the real-world distinctions between members and non-members. To address this, we constructed an authentic and verifiable benchmark in which members are traceable to the public datasets declared by the evaluated models, and non-members are created after the release cut-off date of the LLMs. Table 2 presents the statistics of our collected benchmark on the Python language. Specifically, we collected member and non-member samples as follows.

Members. We leveraged the Pile (Gao et al. 2020), a large-scale open-source dataset composed of multiple smaller datasets, including 7.6% of a repository dataset sourced from GitHub (Gao et al. 2025). The Pile was released in 2021, serving as the training dataset for many LLMs, including Pythia (Biderman et al. 2023), GPT-Neo (Black et al. 2021), StableLM (Stability AI 2023), etc. We randomly sampled 1,000 Python functions by selecting 10 functions from every 100 consecutive entries in the Pile dataset.

Non-members. For non-members, we searched GitHub using a customized query for Python repositories, limiting the creation time to after January 1, 2024 (All four evaluated LLMs had been released prior to this date.). We sorted the repositories by star count in descending order to ensure quality. We then extracted 10 Python functions from each repository before proceeding to the next one, continuing this process across 100 repositories to collect a total of 1,000 functions. To ensure these functions were genuinely original and not cloned from pre-existing sources, we implemented a rigorous verification process. First, we parsed each candidate function's code using Python's `ast` module to extract its name, variable names, and function calls. These elements were then used to build search queries for the GitHub API. The verification relied on three heuristics: (1) searching for the exact function name to identify direct duplicates; (2) searching by internal variable names to detect refactored code reuse; and (3) searching for the complete string of function calls to find logic similarities. Two of the authors conducted peer reviews on the search results to ensure that all 1,000 functions were original and created after January 2024.

Label	Files (#)	Func. (#)	Aver. LOC (#)
Members	-	1000	14.03
Non-Members	214	1000	25.34

Table 2: Benchmark Statistics

Category	Count
Data Model	63,594
Expressions	30,988
Single Statements	321
Compound Statements	11,816
Total Syntax Tokens:	95,257
Total Tokens:	248,218
Syntax Tokens Ratio:	38.4%

Table 3: Syntax Convention Counts in our Benchmark

Table 2 summarizes the statistics of our collected benchmark. The benchmark includes 214 non-member function files with an average of 25.34 lines of code (LOC). For member functions, file counts are unavailable as this information was not provided in the Pile dataset.

We count the occurrences of the syntax conventions (*i.e.,* conditions and consequences) that existed in our benchmark. Table 3 presents the distribution of these syntax convention categories in our benchmark, which demonstrates the syntactic diversity and complexity of the collected functions. The data model accounts for the largest portion of syntax convention tokens, contributing 63,594 tokens, followed by expressions with 30,988 tokens. In total, the four categories result in 95,257 tokens that would be pruned by SYNPRUNE for MIA, representing 38.4% of the overall token set.

Ratio Settings. We evaluated the MIAs under three different member-to-non-member ratios: 1:1, 1:5, and 5:1. For the 1:1 setting, we used 1,000 members and 1,000 non-members. For the 1:5 setting, we randomly sampled 200 members and combined them with 1,000 non-members. Conversely, the 5:1 setting consisted of 1,000 members and a random sample of 200 non-members. These varied distributions were designed to assess the performance and robustness of performing MIAs of code members in LLMs, particularly under imbalanced dataset scenarios.

Experiments

Setup

Baselines. We replicate four recent and representative baselines that represent the current state-of-the-art.

- LOSS (Yeom et al. 2018). LOSS uses the overall perplexity (i.e., cross-entropy loss) of a language model as the detection score, based on the standard assumption in mem-

Ratio	Method	Pythia 2.8B	GPT-Neo 2.7B	StableLM-Alpha 3B	GPT-J 6B
1:1	LOSS	38.4	43.7	33.3	43.3
	ZLIB	33.2	35.5	32.0	35.7
	MIN-K	38.8	41.8	34.2	41.1
	DC-PDD	50.1	41.2	43.7	41.3
	SYNPRUNE	61.3	59.7	61.3	59.7
1:5	LOSS	40.3	45.3	33.9	44.8
	ZLIB	34.0	36.2	32.5	36.3
	MIN-K	40.2	43.3	34.9	42.4
	DC-PDD	51.8	44.0	44.6	43.2
	SYNPRUNE	61.2	59.4	61.1	61.5
5:1	LOSS	40.0	43.8	34.9	43.6
	ZLIB	33.1	35.0	31.9	35.2
	MIN-K	39.9	42.2	35.1	41.6
	DC-PDD	49.4	40.9	44.1	39.6
	SYNPRUNE	62.0	60.7	62.0	63.1

Table 4: AUROC (%) of Different Methods under Varying Member-to-non-member Ratios

bership inference attacks that training data yields lower perplexity than unseen data.

- ZLIB (Carlini et al. 2021). ZLIB applies the ZLib compression algorithm to each sample's tokenized representation, using the compressed length as a detection score.
- MIN-K% (Shi et al. 2023). MIN-K% ranks tokens by likelihood and computes a score using the bottom K based on their aggregated likelihoods.
- DC-PDD (Zhang et al. 2024). The DC-PDD improves performance on pre-training data by calibrating distributional differences, suitable for multilingual scenarios.

Targeted LLMs. We found evidence that the following four models were trained on the Pile dataset: *EleutherAI/pythia-2.8b* (EleutherAI 2025c), *EleutherAI/gpt-neo-2.7B* (EleutherAI 2025b), *StabilityAI/stablelm-base-alpha-3b* (stability ai 2025), and *EleutherAI/gpt-j-6b* (EleutherAI 2025a). Therefore, these four models can be evaluated on member and non-member samples from our constructed benchmark. Specifically, *EleutherAI/gpt-neo-2.7B* is a 2.7B parameter Transformer model designed for text generation and creative tasks. *StabilityAI/stablelm-base-alpha-3b* is a 3B parameter model optimized for code generation and low-resource deployment. *EleutherAI/gpt-j-6b* is a 6B parameter model with enhanced reasoning capabilities.

Metrics. We use the Area Under the Receiver Operating Characteristic curve (AUROC) as our evaluation metric. AUROC is widely adopted for binary classification tasks and is particularly suitable for membership inference. It captures the trade-off between true positive rate and false positive rate, providing an aggregate judge of the effect of all possible thresholds (i.e., ϵ). It offers a robust measure of a method's ability to distinguish between training and non-training data in language models. We also use the false negative rate (FNR) to measure the percentage of members missed by the detection techniques, and the f1-score to select the most effective threshold in different models.

Length	Pythia 2.8B	GPT-Neo 2.7B	StableLM-Alpha 3B	GPT-J 6B
Short	2.10	19.39	0.00	0.00
Long	75.12	75.13	66.04	87.09

Table 5: False Negative Rate (%) of SYNPRUNE on Members under Short and Long Function Lengths

Environment. Our experiments were conducted on a single NVIDIA RTX 4090D GPU (24GB), supported by a 16-core Intel Xeon Platinum 8474C CPU and 80GB of RAM. The software environment consisted of Ubuntu 20.04, Python 3.8, PyTorch 2.0.0, and CUDA 11.8.

Results

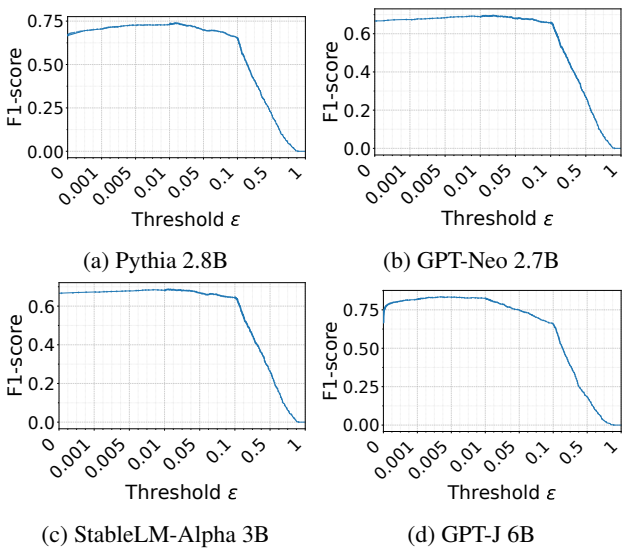
Table 4 presents the effectiveness of SYNPRUNE compared to baseline methods across four models, demonstrating consistent superiority in 1:1, 1:5, and 5:1 ratios. Specifically, SYNPRUNE achieves average AUROCs of 61.5%, 59.9%, 61.4%, and 61.4% in Pythia 2.8B, GPT-Neo 2.7B, StableLM-Alpha 3B, and GPT-J 6B, respectively. SYNPRUNE maintains consistent and similar effectiveness across difference model architectures and scales. Compared to state-of-the-art methods, SYNPRUNE achieves AUROC improvements ranging from 11.2% to 17.6% (1:1 ratio), 9.4% to 16.7% (1:5 ratio), and 12.6% to 19.5% (5:1 ratio), underscoring its robust performance advantages. Overall, SYNPRUNE achieves an average AUROC improvement of 15.4% across four models and three data ratios compared to the state-of-the-art.

Robustness under Different Function Lengths

We categorize the member functions by token lengths into two groups based on statistical distribution. Specifically, we sort all member functions in the benchmark by their token

Ratio	Method	Pythia 2.8B	GPT-Neo 2.7B	StableLM-Alpha 3B	GPT-J 6B
1:1	w/o DM	↓5.1	↓2.9	↓5.8	↓1.8
	w/o Expr.	↓4.5	↓3.6	↓5.6	↓2.4
	w/o S.Stmt	↓5.0	↓2.5	↓6.0	↓1.6
	w/o C.Stmt	↓6.1	↓6.7	↓7.1	↓3.8
1:5	w/o DM	↓4.1	↓0.1	↓4.7	↓1.0
	w/o Expr.	↓3.1	↓2.6	↓4.5	↓3.1
	w/o S.Stmt	↓4.4	↓2.2	↓5.1	↓2.6
	w/o C.Stmt	↓4.8	↓6.5	↓6.2	↓4.9
5:1	w/o DM	↓2.7	↓0.1	↓3.6	↓1.2
	w/o Expr.	↓2.6	↓4.4	↓4.1	↓3.0
	w/o S.Stmt	↓3.2	↓0.7	↓3.9	↓1.8
	w/o C.Stmt	↓4.1	↓3.2	↓5.2	↓3.9

Table 6: Ablation Study (Each cell indicates the difference in AUROC compared to the original SYNPRUNE.)



(a) Pythia 2.8B (b) GPT-Neo 2.7B

(c) StableLM-Alpha 3B (d) GPT-J 6B

Figure 2: F1-Score of SYNPRUNE Across Varying Thresholds

counts in ascending order, then divide them into two groups (500:500) using the median number as the split point: *short* (token count ≤ 55) and *long* (token count > 55). This balanced distribution ensures fair comparison under varying function complexities.

Table 5 shows SYNPRUNE's effectiveness in terms of false negative rate. The results demonstrate that SYNPRUNE generates fewer false positives with shorter functions. Notably, SYNPRUNE achieved perfect member recall when evaluated on both StableLM-Alpha 3B and GPT-J 6B, correctly identifying all members without any misses. Meanwhile, longer functions correlate with increased false negative rates, indicating future MIA improvements in handling longer functions.

Ablation Study

We created four ablated versions of SYNPRUNE by individually removing syntax conventions by categories: data model (w/o DM), expressions (w/o Expr.), single statements (w/o S.Stmt), and compound statements (w/o C.Stmt).

We evaluated the four ablated versions on the three ratios, with the results presented in Fig. 6. Each cell shows the difference in AUROC compared to the original SYNPRUNE. We observe that all ablated versions exhibit an AUROC drop across all three ratios, demonstrating the contribution of the syntax conventions in each category. Our ablation study reveals the impact of syntax convention removal on model performance. Eliminating data model syntax conventions resulted in average AUROC decreases of 3.9% (1:1), 2.5% (1:5), and 1.9% (5:1). Similarly, removing expression syntax conventions led to reductions of 4.0%, 3.3%, and 3.5%, while ablating single statement conventions caused drops of 3.8%, 3.6%, and 2.4% across the respective ratios. The most significant impact came from compound statement syntax removal, with AUROC losses of 5.9%, 5.6%, and 4.1%.

Threshold Analysis

Fig. 2 presents the curve of f1-score of SYNPRUNE in the four models across varying thresholds. Note that we applied multiple axis scales to better illustrate the results, as the SPP scores are typically very small. Observing from the graph, we can see that all four models undergo an minor increase of F1-score when ϵ increases and confront a plunge after ϵ exceeds around 0.01. We selected the most effective ϵ for the four models. Specifically, Pythia achieves an F1-score of 0.7390 at $\epsilon = 0.0175$; GPT-Neo reaches 0.6963 at $\epsilon = 0.0250$; StableLM-Alpha records 0.6966 at $\epsilon = 0.0143$; and GPT-J achieves 0.8432 at $\epsilon = 0.0037$.

Limitations

While SYNPRUNE is currently implemented and evaluated using Python's grammar, its design readily extends to other languages (e.g., Java, C/C++, JavaScript). This generalizability stems from shared language conventions and the universal requirement of strict grammatical correctness across programming languages. We therefore anticipate SYNPRUNE would achieve similar effectiveness when applied to these other languages. Besides, our evaluation is currently limited to four LLMs due to resource constraints and limited transparency regarding other LLMs' training data. Nevertheless, the consistent results across all models suggest strong generalizability of our approach to other large language models.

Conclusions

Identifying the inclusion of specific code samples in LLM training data is crucial for enhancing transparency, supporting accountability, and ensuring copyright compliance. We introduced SYNPRUNE, a syntax-pruned MIA technique specifically tailored for detecting pretraining code members from LLMs. SYNPRUNE focuses on more semantically meaningful signals of memorization. We evaluate SYNPRUNE on a real-world benchmark of Python functions that distinguishes verifiable members from non-members, showing that SYNPRUNE consistently outperforms existing MIA baselines.

Ethics Statement

This work is motivated by the pressing concern of respecting intellectual property in the development of large language models (LLMs), particularly regarding the use of copyrighted code in pretraining datasets. While membership inference attacks (MIAs) raise privacy and model extraction concerns, we limit our use to research contexts aimed at evaluating potential copyright risks, not for exploitation. We adhere to responsible disclosure norms and aim to foster greater transparency and accountability in LLM development.

Acknowledgements

This work was supported in part by the National Key R&D Program of China (Grant No. 2023YFC3806000 and 2023YFC3806002), in part by the National Natural Science Foundation of China (Grant No. 62402342, 62402113), in part by Shanghai Municipal Science and Technology Major Project No. 2021SHZDZX0100, and in part by Shanghai Sailing Program (No. 24YF2749500). The authors would also like to thank the anonymous reviewers for their careful work and valuable suggestions.

References

Biderman, S.; Schoelkopf, H.; Anthony, Q.; et al. 2023. Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. *arXiv preprint arXiv:2304.01373*.

Black, S.; Gao, L.; Wang, P.; Leahy, C.; and Biderman, S. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. https://www.eleuther.ai/projects/gpt-neo/.

Carlini, N.; Chien, S.; Nasr, M.; Song, S.; Terzis, A.; and Tramer, F. 2022. Membership inference attacks from first principles. In *2022 IEEE symposium on security and privacy (SP)*, 1897–1914. IEEE.

Carlini, N.; Tramer, F.; Wallace, E.; Jagielski, M.; Herbert-Voss, A.; Lee, K.; Roberts, A.; Brown, T.; Song, D.; Erlingsson, U.; et al. 2021. Extracting training data from large language models. In *30th USENIX security symposium (USENIX Security 21)*, 2633–2650.

Chen, T.; Asai, A.; Mireshghallah, N.; Min, S.; Grimmelmann, J.; Choi, Y.; Hajishirzi, H.; Zettlemoyer, L.; and Koh, P. W. 2024. CopyBench: Measuring literal and non-literal reproduction of copyright-protected text in language model generation. *arXiv preprint arXiv:2407.07087*.

Choquette-Choo, C. A.; Tramer, F.; Carlini, N.; and Papernot, N. 2021. Label-only membership inference attacks. In *International conference on machine learning*, 1964–1974. PMLR.

Duan, M.; Suri, A.; Mireshghallah, N.; Min, S.; Shi, W.; Zettlemoyer, L.; Tsvetkov, Y.; Choi, Y.; Evans, D.; and Hajishirzi, H. 2024. Do membership inference attacks work on large language models? *arXiv preprint arXiv:2402.07841*.

EleutherAI. 2025a. gpt-j-6b. https://huggingface.co/EleutherAI/gpt-j-6b. Accessed: 2025-12-06.

EleutherAI. 2025b. GPT-Neo 2.7B. https://huggingface.co/EleutherAI/gpt-neo-2.7B. Accessed: 2025-12-06.

EleutherAI. 2025c. pythia-2.8b-v0. https://huggingface.co/EleutherAI/pythia-2.8b-v0. Accessed: 2025-12-06.

Foundation, F. S. 2025a. GNU Software. https://www.gnu.org/software/software.html. Accessed: 2025-12-06.

Foundation, P. S. 2025b. Python 3.11.12 documentation. https://docs.python.org/3.11/. Accessed: 2025-12-06.

Gao, L.; Biderman, S.; Black, S.; Golding, L.; Hoppe, T.; Foster, C.; Phang, J.; He, H.; Thite, A.; Nabeshima, N.; Presser, S.; and Leahy, C. 2020. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*.

Gao, L.; Biderman, S.; Black, S.; Golding, L.; Hoppe, T.; Foster, C.; Phang, J.; He, H.; Thite, A.; Nabeshima, N.; Presser, S.; and Leahy, C. 2025. The Pile.

Grynbaum, M. M.; and Mac, R. 2025. The Times sues OpenAI and Microsoft over A.I. use of copyrighted work.

He, Y.; Li, B.; Wang, Y.; Yang, M.; Wang, J.; Hu, H.; and Zhao, X. 2024. Is Difficulty Calibration All We Need? Towards More Practical Membership Inference Attacks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 1226–1240.

Huang, Z.; Gong, N. Z.; and Reiter, M. K. 2024. A General Framework for Data-Use Auditing of ML Models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 1300–1314.

Informationweek. 2024. First lawsuit over GPL settled.

Kazmi, M.; Lautraite, H.; Akbari, A.; Tang, Q.; Soroco, M.; Wang, T.; Gambs, S.; and Lécuyer, M. 2024. Panoramia: Privacy auditing of machine learning models without retraining. *Advances in Neural Information Processing Systems*, 37: 57262–57300.

Leino, K.; and Fredrikson, M. 2020. Stolen memories: Leveraging model memorization for calibrated {White-Box} membership inference. In *29th USENIX security symposium (USENIX Security 20)*, 1605–1622.

Li, J.; Shen, Y.; Huang, S.; Dai, X.; and Chen, J. 2021. When is char better than subword: A systematic study of segmentation algorithms for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 543–549.

Liu, X.; Sun, T.; Xu, T.; Wu, F.; Wang, C.; Wang, X.; and Gao, J. 2024. Shield: Evaluation and defense strategies for copyright compliance in llm text generation. *arXiv preprint arXiv:2406.12975*.

Liu, Y.; Zhao, Z.; Backes, M.; and Zhang, Y. 2022. Membership inference attacks by exploiting loss trajectory. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2085–2098.

LLP, M. L. 2025. Developers Sue GitHub, Microsoft, and OpenAI Over Copyright in Creating AI Tool Copilot.

Long, Y.; Bindschaedler, V.; Wang, L.; Bu, D.; Wang, X.; Tang, H.; Gunter, C. A.; and Chen, K. 2018. Understanding membership inferences on well-generalized learning models. *arXiv preprint arXiv:1802.04889*.

Lu, S.; Guo, D.; Ren, S.; Huang, J.; Svyatkovskiy, A.; Blanco, A.; Clement, C.; Drain, D.; Jiang, D.; Tang, D.; et al. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

lvcriminaldefense. 2024. First open source GPL violation lawsuit.

Mattern, J.; Mireshghallah, F.; Jin, Z.; Schölkopf, B.; Sachan, M.; and Berg-Kirkpatrick, T. 2023. Membership inference attacks against language models via neighbourhood comparison. *arXiv preprint arXiv:2305.18462*.

Meeus, M.; Shilov, I.; Faysse, M.; and De Montjoye, Y.-A. 2024. Copyright traps for large language models. *ICML 2024*.

Nasr, M.; Shokri, R.; and Houmansadr, A. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, 739–753. IEEE.

Rezaei, S.; and Liu, X. 2021. On the difficulty of membership inference attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7892–7900.

Sablayrolles, A.; Douze, M.; Schmid, C.; Ollivier, Y.; and Jégou, H. 2019. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning*, 5558–5567. PMLR.

Salem, A.; Zhang, Y.; Humbert, M.; Berrang, P.; Fritz, M.; and Backes, M. 2018. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*.

Shi, W.; Ajith, A.; Xia, M.; Huang, Y.; Liu, D.; Blevins, T.; Chen, D.; and Zettlemoyer, L. 2023. Detecting pretraining data from large language models. *ICLR 2024*.

Shibata, Y.; Kida, T.; Fukamachi, S.; Takeda, M.; Shinohara, A.; Shinohara, T.; and Arikawa, S. 1999. Byte pair encoding: A text compression scheme that accelerates pattern matching.

Shilov, I.; Meeus, M.; and de Montjoye, Y.-A. 2024. Mosaic memory: Fuzzy duplication in copyright traps for large language models. *arXiv preprint arXiv:2405.15523*.

Shokri, R.; Stronati, M.; Song, C.; and Shmatikov, V. 2017. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, 3–18. IEEE.

Song, C.; and Shmatikov, V. 2019. Auditing data provenance in text-generation models. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 196–206.

Song, L.; and Mittal, P. 2021. Systematic evaluation of privacy risks of machine learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, 2615–2632.

Song, L.; Shokri, R.; and Mittal, P. 2019. Privacy risks of securing machine learning models against adversarial examples. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 241–257.

Stability AI. 2023. StableLM-Base-Alpha-3B: An Open Source 3B Parameter Language Model. https://huggingface.co/stabilityai/stablelm-base-alpha-3b.

stability ai. 2025. stablelm-base-alpha-3b. https://replicate.com/stability-ai/stablelm-base-alpha-3b/readme. Accessed: 2025-12-06.

Tirumala, K.; Markosyan, A.; Zettlemoyer, L.; and Aghajanyan, A. 2022. Memorization without overfitting: Analyzing the training dynamics of large language models. *Advances in Neural Information Processing Systems*, 35: 38274–38290.

Truex, S.; Liu, L.; Gursoy, M. E.; Yu, L.; and Wei, W. 2019. Demystifying membership inference attacks in machine learning as a service. *IEEE transactions on services computing*, 14(6): 2073–2089.

Vynck, G. D. 2025. 8 major newspapers join legal backlash against OpenAI, Microsoft.

Wan, Y.; Wan, G.; Zhang, S.; Zhang, H.; Zhou, P.; Jin, H.; and Sun, L. 2024. Does your neural code completion model use my code? a membership inference approach. *arXiv preprint arXiv:2404.14296*.

Watson, L.; Guo, C.; Cormode, G.; and Sablayrolles, A. 2021. On the importance of difficulty calibration in membership inference attacks. *arXiv preprint arXiv:2111.08440*.

wikipedia. 2024. Open source license litigation.

Xu, J.; Li, S.; Xu, Z.; and Zhang, D. 2024. Do LLMs Know to Respect Copyright Notice? *EMNLP 2024*.

Yang, Z.; Zhao, Z.; Wang, C.; Shi, J.; Kim, D.; Han, D.; and Lo, D. 2024. Gotcha! this model uses my code! evaluating membership leakage risks in code models. *IEEE Transactions on Software Engineering*.

Ye, J.; Maddi, A.; Murakonda, S. K.; Bindschaedler, V.; and Shokri, R. 2022. Enhanced membership inference attacks against machine learning models. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 3093–3106.

Yeom, S.; Giacomelli, I.; Fredrikson, M.; and Jha, S. 2018. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, 268–282. IEEE.

Zhang, W.; Zhang, R.; Guo, J.; de Rijke, M.; Fan, Y.; and Cheng, X. 2024. Pretraining data detection for large language models: A divergence-based calibration method. *arXiv preprint arXiv:2409.14781*.

Zhou, B.; Wang, Z.; Wang, L.; Wang, H.; Zhang, Y.; Song, K.; Sui, X.; and Wong, K.-F. 2024. DPDLLM: A Black-box Framework for Detecting Pre-training Data from Large Language Models. In *Findings of the Association for Computational Linguistics ACL 2024*, 644–653.