# Similarity detection on LLM generated source code with SCANOSS

PhD. Oscar Enrique Goñi

February 27, 2025

**Abstract**

This paper investigates source code similarity detection in Large Language Model (LLM) outputs using the SCANOSS platform. While recent research has identified concerns regarding LLMs generating code that closely resembles their training data, the full extent of this similarity across the broader open-source ecosystem remains unexplored. We extend previous research by evaluating generated code snippets against SCANOSS's comprehensive open-source knowledge base, which provides a more complete assessment of potential code similarity beyond the original training datasets. We have used SCANOSS's Winnowing-based scanning algorithm as starting point to analyze the original metrics described in [XGHZ24a], and also defined a new metric to evaluate code snippets at various similarity thresholds. Our findings reveal that approximately 30% of analyzed code exhibits at least 10% similarity to existing open-source implementations, while 1% maintains similarity even at a more stringent 30% threshold. These results indicate that code similarity in LLM outputs may be more prevalent than previously indicated when evaluated against a broader open-source codebase. Our study contributes to the ongoing discussion of LLM-generated code's originality and its implications for software licensing compliance, while validating the effectiveness of lightweight similarity detection algorithms as preliminary indicators for more comprehensive analysis

## 1 Introduction

In recent years, the relentless global interest in AI has led to the emergence of numerous Large Language Models (LLMs). As a product of extensive human research, AI promises to be a transformative tool that will complement future productive activities.

Software development stands as one of many fields already embracing this technology, particularly in source code generation based on programmer intent (prompts). However, since AI models acquire training through examples, the generated source code often exhibits significant similarity to, or exact replication of, their training data. While this poses no inherent limitation when the original source code carries permissive licensing, it can lead to serious legal implications when dealing with incompatible licenses.

## 2 Background

In 2023, SCANOSS began receiving inquiries regarding their platform's capability to detect LLM-generated source code. This led to an initial experiment with the GPT model, where a prompt was designed to implement a specific algorithm used in SCANOSS's own platform.

The third-generation output produced an exact SCANOSS implementation containing:

- Identical comments
- Identical variable names
- Identical syntactic structure
- Minor variations in comment formatting (substitution of `/* */` with `//`)
- Variations in method/function definition ordering

### 2.1 Related work

Research conducted in [XGHZ24a] addresses this challenge through an empirical study to identify a reasonable standard of *"notable similarity"* that excludes the possibility of independent creation, indicating a copying relationship between LLM output and specific open-source code. Based on this standard, the authors propose LICOEVAL [XGHZ24b], an evaluation benchmark for measuring LLMs' license compliance capabilities. Their evaluation of 14 popular LLMs revealed that even top-performing models produce a non-negligible proportion (0.88% to 2.01%) of code notably similar to existing open-source implementations. Significantly, most LLMs fail to provide accurate license information, particularly for code under copyleft licenses.

## 3 Hypothesis

While the analyzed work considers a substantial dataset (783GB of code across 86 languages, with 74,772,489 Python functions initially extracted and filtered to 2,628,395 based on specific criteria), the results are analyzed on a subset of the universe of Open Source Code. This report aims to evaluate the original study's findings against a broader Open Source Code base.

## 4 Experimental setup

Although access to the original prompts and complete detailed results was limited, we obtained generated code samples from GitHub. We were able to collect 10,000 source code snippets generated by the original work. This experiment uses the SCANOSS knowledge-base through the **osskb.org** service provided by the **Software Transparency Foundation**[stf25] and running SCANOSS's open-source scanning tools [SCA25].

The scanning platform provides three types of results:

- *No Match*: No coincidences found.
- *Snippet Match*: Coincidences found in certain code portions
- *File Match*: Exact match found

Given that the analysis involves isolated functions rather than complete files, *File Match* results are not expected.

As of this white-paper's publication, the SCANOSS knowledge base encompasses 27 terabytes of unique open-source software (OSS) code, sourced from more than 250 million URLs. With a monthly growth rate of approximately 2%, this represents a search space approximately 35 times larger than the original dataset.

### 4.1 Technical Overview: SCANOSS Snippet Matching Algorithm

SCANOSS platform implements a *Winnowing* mechanism to detect equivalent portions of the source code [SCA20]. During scanning initiation, the CLI [sp20] converts each line of source code into sets of fingerprints for searching and matching. This approach prevents users from sending actual source code content and makes reconstruction virtually impossible. Additionally, the service is stateless, leaving no trace of any queries. Using the fingerprint set, the scanning engine searches for matches within the Knowledge Base, providing information about the origin of matching code and corresponding line ranges. Although the algorithm effectively ignores minor formatting changes, it may not detect matches where variables have been renamed or values replaced with constants. For cases demanding precise line-by-line comparison, the SCANOSS engine implements High Precision Snippet Matching (HPSM). This advanced feature necessitates the transmission of supplementary hashes that represent the vertical encoding of the source code structure, enabling more granular matching capabilities.

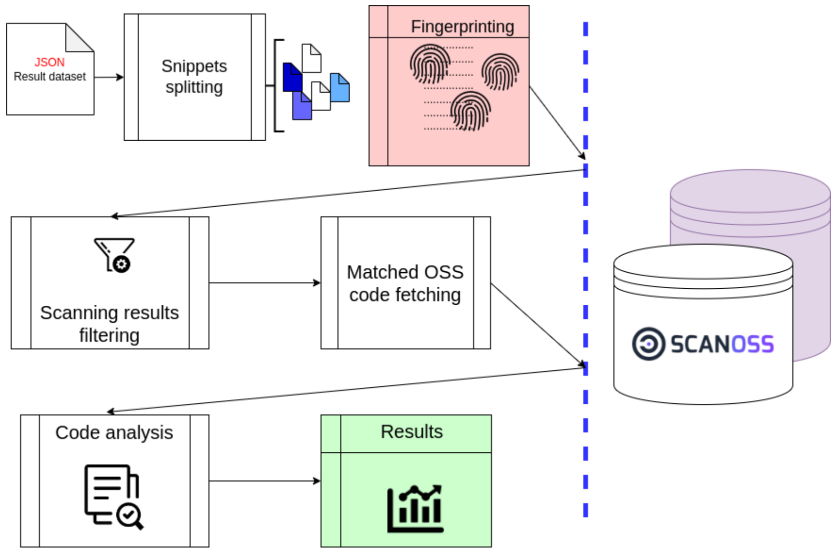

Figure 1: Experiment setup

## 5 Methodolgy

The original research implements a comprehensive categorization system for source code snippets. However, for the purposes of our investigation, we have concentrated solely on leveraging the data set constructed in the original work, without modifying its classification structure.

1. **Snippet creation**: Our methodological approach begins with the disaggregation of dataset snippets into discrete source code files. This preliminary decomposition is a prerequisite for executing the initial scanning process against the OSSKB.org knowledge base.

2. **Fingerprinting**: Each file is subjected to a fingerprinting procedure and transmitted to the **OSSKB.org** service. This implementation utilizes the **scanoss-py** tool with default parameters. The tool performs two key functions: *automatic fingerprint generation* and *submission for comparative analysis.*

3. **Scanning**: Once received by the service, the scanning engine executes a search algorithm to identify the minimum acceptable match threshold. Upon completion of the fingerprint comparison process, the system returns comprehensive results to the client in JSON format, encompassing detailed information about both the matched files and their associated components.

4. **Filtering**: The scan results are used as initial indicators to facilitate a comprehensive analysis of the identified matching source code segments. This preliminary matching serves as a foundation for a more detailed comparative examination. This work focuses only on the following data of interest from the scanning results:

   - File Hash: The MD5 of the OSS file that matches. This value will be used as a key to fetch the OSS source code content.
   - Match type: Snippet matching is expected where coincidences are found. A value equal to none means that no coincidences were found.
   - Match range: Scanned snippet and OSS lines that match. We anticipate identifying a limited matching range that is proportional to and constrained by the total length of the analyzed snippet.

   The algorithm's characteristics mentioned above do not present significant limitations for our study. The detection of even small matching code portions provides sufficient starting points for more detailed comparison. In this particular case, the HPSM feature is kept disabled during scanning. Lets consider the following source code snippets

```
// Snippet A
for (int i=0; i<10; i++){
    acum += i*10
}
```

```
// Snippet B
for (int i=0; i<10; i++)
{
  acum += i*10
}
```

While Snippets A and B share identical semantics and generate identical fingerprints due to their textual similarity, the variation in block opening style results in a different vertical fingerprint pattern, causing the system to discard what would otherwise be a valid match. In the particular context of this investigation, the principal justification for disabling the feature lies in the observation that strict exact matching might exclude potentially relevant matches, while more flexible matching criteria can establish foundational points for a deeper analytical examination.

5. **Evaluation**: Upon identification of potential OSS matches, we initiate an in-depth analytical process of the potentially corresponding code snippets. This analytical phase commences with the retrieval of complete source content from each identified OSS file. Once in place, the following metrics are evaluated using both OSS file and the generated snippet:

   - Jaccard distance
   - Shared shingles
   - Bleu4
   - Winnowing match ratio

   For the purpose of conducting an equitable comparative analysis with the original investigation's results, we implement the same set of metrics and preserve the original matching code proportion. This methodological consistency ensures the validity of our comparative analysis.

   Furthermore, the analysis has been enhanced by incorporating a Winnowing Matching Rate (WMR). This metric measures the proportion between the matching OSS code size and the generated code size, providing insight into the relative scale of the identified matches. The primary objective of this additional measurement is to assess the scanning engine's performance and response characteristics when processing these specific code snippet types.

$$WMR = \frac{\#MatchedOSSlines}{\#GeneratedSnippetLines}$$

## 6 Results

Our experimental methodology employs a one-shot execution utilizing an automated script, following the process detailed in 5. The experimental results, presented in 1, demonstrate the relationship between varying similarity thresholds and match rates. Analysis of the data reveals two significant findings: at the most permissive similarity threshold of 5%, approximately 30% of the analyzed source code corresponds to OSS files. In contrast, when applying the most stringent similarity threshold of 30%, the match rate maintains a level exceeding 1%, indicating persistent code similarity even under restrictive conditions.

| Threshold | Jaccard distance | Shared Shringles | Bleu4 | Winnowing |
|---|---|---|---|---|
| 10 | 543 | 528 | 2019 | 3515 |
| 15 | 272 | 257 | 1408 | 3275 |
| 20 | 152 | 137 | 1006 | 2873 |
| 25 | 78 | 78 | 652 | 2433 |
| 30 | 48 | 49 | 428 | 2068 |

Table 1: Similarity vs. number of source code snippets matching

The analysis encompasses the entire source code text, including both executable code and comments. Given that comments within the code may have undergone translation or reformulation, it is important to note that these modifications could have influenced the calculated similarity metrics. This consideration warrants attention when interpreting the overall similarity assessments.

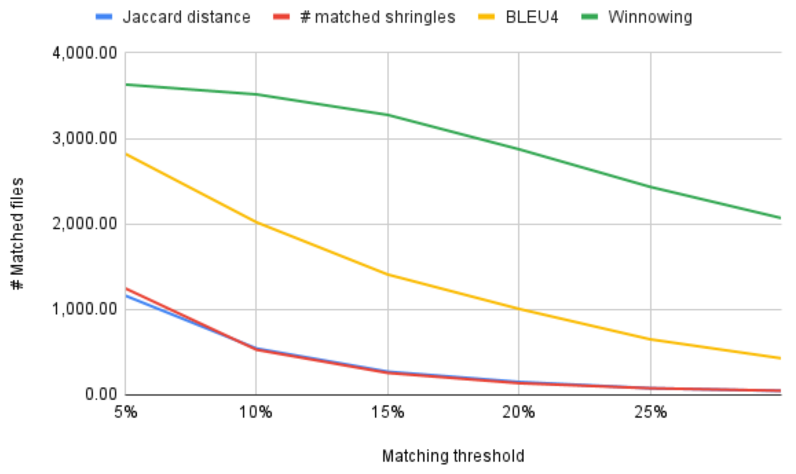

Figure 2: Evaluation of matching ranges for different metrics

Observation of the data indicates a consistent trend across all metrics, with both BLEU4 and Winnowing algorithms displaying notably more permissive characteristics in their similarity assessments compared to other measures.

| Threshold | Jaccard distance | Bleu4 | Winnowing |
|---|---|---|---|
| 10-30% | 0.5-5.7% | 4.28-20.1% | 20.6-35.1% |

Table 2: Evaluation of similarity of LLM generated code vs. OSS code for the proposed metrics

The Table 2 presents a consolidated analysis of similarity metrics evaluated across the complete dataset consisting of 10,000 generated code snippets, providing quantitative measurements for each applied comparison methodology.

## 7 Conclusions

This work describes an experiment designed to validate the findings of previous research concerning the similarity between LLM-generated source code and the original source code used in training. While our study did not focus on code generation itself, it utilizes a subset of code generated from the previous investigation as a reference point.

We incorporated the metrics from the original study and supplemented them with SCANOSS's Winnowing algorithm metric (WMR). Our findings support the original research conclusions while indicating a higher upper detection threshold. The original work reported matches ranging from 0.8% to 2%, whereas our experiment yielded values between 0.5% and 5.3%. Although we considered a smaller input set, our analysis encompassed a larger knowledge base, and we attribute the difference between both studies to this expanded scope. Figure 3 depicts the visual comparison between both works.

Notably, the behavior of the Winnowing metric reported by the SCANOSS search engine proves particularly interesting. While the values produced by this metric alone would not definitively determine whether code was LLM-generated, its trend aligned consistently with other more complex metrics. This observation validates our hypothesis that Winnowing results can serve as an effective starting point for more detailed analysis. The research demonstrates that expanding the reference knowledge base reveals higher similarity rates than previously identified, while confirming the fundamental findings of the original study. Additionally, it validates the effectiveness of simpler matching algorithms as preliminary indicators for more comprehensive analysis.

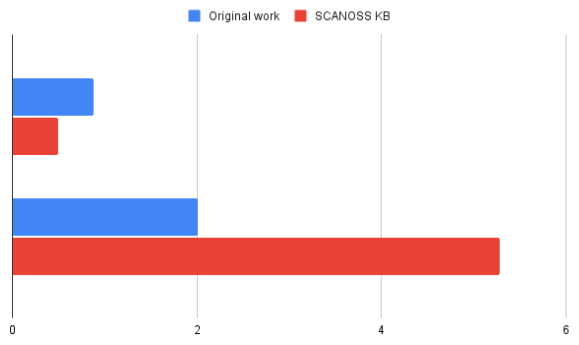

Figure 3: Results comparison setup

## 8 Discussion

Analysis of the similarity metrics reveals that both BLEU4 and Winnowing consistently report higher similarity rates compared to Jaccard distance. This disparity can be attributed to their inherent ability to capture structural and syntactic patterns in source code. While Jaccard distance operates primarily on direct text comparison, BLEU4 and Winnowing employ more sophisticated mechanisms that can detect functional similarities even when surface-level modifications, such as variable renaming or formatting changes, are present. BLEU4's n-gram based approach allows it to recognize repeated code patterns and structural similarities while being less sensitive to localized changes. Similarly, Winnowing's fingerprinting mechanism is designed to identify core algorithmic patterns while being resilient to common code modifications. These characteristics make both metrics particularly well-suited for source code similarity analysis, as they better reflect the functional equivalence of code segments rather than merely their textual similarity. The higher detection rates of these metrics should not be interpreted as false positives, but rather as a more comprehensive capture of actual code similarity. In the context of source code analysis, where functional equivalence is often more relevant than exact textual matching, BLEU4 and Winnowing provide more meaningful similarity assessments than traditional text-based metrics like Jaccard distance.

## References

[SCA20] SCANOSS. Source code fingerprinting with winnowing. https://github.com/scanoss/wfp, 2020. Accesed: 10-02-2024.

[SCA25] SCANOSS. Open source inventory engine built for modern devsecops. https://github.com/scanoss, 2020-2025. Accesed: 10-02-2024.

[sp20] scanoss py. Scanoss scanning cli. https://github.com/scanoss/scanoss-py, 2020. Accesed: 10-02-2024.

[stf25] stf. Software transparency foundation. https://st.foundation, 2020-2025. Accesed: 10-02-2024.

[XGHZ24a] Weiwei Xu, Kai Gao, Hao He, and Minghui Zhou. A first look at license compliance capability of llms in code generation, 08 2024.

[XGHZ24b] Weiwei Xu, Kai Gao, Hao He, and Minghui Zhou. Licoeval: Evaluating llms on license compliance in code generation, 2024.