Project 8: Memorization of public SE datasets in LLMs

Omar Dinari
University of L'Aquila, Italy
omar.dinari@student.univaq.it

Alessandro Di Giacomo
University of L'Aquila, Italy
alessandro.digiacomo@student.univaq.it

Agostino D'Agostino
University of L'Aquila, Italy
agostino.dagostino@student.univaq.it

Course: Machine Learning for Software Engineering (DT1052)
Prof. Dr. Phuong Nguyen
University of L'Aquila, Italy

Academic Year 2025-2026

Abstract

Large Language Models (LLMs) such as Qwen, Mistral, and Llama have revolutionized automated code generation. However, their efficacy raises concerns regarding "memorization", the tendency of models to reproduce training data verbatim rather than generalizing to new problems. This project implements a probing framework to quantify memorization in LLMs using the CodeSearchNet dataset. We employ a methodology based on docstring perturbation and MinHash similarity metrics. Our implementation features a modular architecture that supports both in-memory and streaming data persistence. Comprehensive sensitivity analysis ($N = 5$ to $N = 100$) and adversarial stress testing reveal that while models like Mistral-7B demonstrate high structural similarity (0.148), they maintain a 0.00% Exact Match rate. This suggests that modern instruction-tuned models exhibit robust generalization rather than overfitting to specific training examples.

Contents

1 Introduction

1.1 Context and Motivation

In recent years, the intersection of Machine Learning (ML) and Software Engineering (SE) has seen rapid advancement driven by Large Language Models (LLMs). Models like Llama and Mistral demonstrate remarkable capabilities in code completion. However, a critical question remains: *Do these models actually understand software engineering principles, or do they simply memorize their training data?*

This distinction is crucial because memorization can lead to **Generalization Failure** (failing on novel problems) and **Data Leakage** (exposing confidential code).

1.2 Project Objectives

This project, aligned with *Project 8: Memorization of public SE datasets in LLMs* [1], aims to:

- Develop a modular Python framework to probe LLMs for memorization.
- Implement metrics such as MinHash Similarity and Exact Match.
- Evaluate robustness by perturbing input prompts.
- Conduct a sensitivity analysis and adversarial testing to validate statistical significance.

2 Methodology

2.1 Metrics for Code Similarity

2.1.1 Jaccard Similarity and MinHash

To measure overlap between generated code and reference code, we use Jaccard Similarity. Since exact calculation is expensive for large datasets, we use **MinHash** (Theta Sketch), which efficiently estimates the overlap of token sets [2].

2.1.2 Robustness Drop

We distinguish memorization from understanding using the "Robustness Drop" (ΔR) metric:

$$\Delta R = S_{orig} - S_{pert} \tag{1}$$

Where S_{orig} is the similarity score with the original docstring, and S_{pert} is the score with a perturbed (paraphrased) docstring. A high ΔR suggests the model was "triggered" by the exact phrasing (memorization).

2.2 Statistical Rigor and Reproducibility

To ensure that our results are scientifically valid and not merely artifacts of randomness, we implemented a strict control strategy.

2.2.1 Controlled Randomness

Machine Learning experiments often rely on stochastic processes. Without control, two runs could yield different results simply because they processed easier or harder samples. We fixed the global random seed to `42`. This guarantees **deterministic shuffling**, ensuring that when we compare Qwen vs. Mistral, both models are evaluated on the **exact same functions**.

2.2.2 Sample Size (N)

We varied the sample size $N \in \{5, 50, 100\}$. In preliminary tests ($N = 5$), we observed high background noise (Baseline ≈ 0.44). By increasing to $N = 100$, the background noise collapsed to ≈ 0.06, providing a reliable baseline for detecting genuine memorization.

3 System Architecture

3.1 Data Persistence Requirement

A key requirement of the project was to ensure flexibility in data management: "The data must be managed both in memory and through files" [1]. We addressed this via the `--streaming` flag:

- **In-Memory Mode:** Loads the full dataset into RAM (Fast access).
- **Streaming Mode:** Streams data on-the-fly from files/network (Low memory usage).

```python
if getattr(args, "streaming", False):
    # Stream from file system/network (Low RAM usage)
    dataset = load_dataset(args.dataset, streaming=True)
    dataset = dataset.shuffle(buffer_size=10000)
else:
    # Load fully into Memory (High RAM usage)
    dataset = load_dataset(args.dataset)
```

Listing 1: Data Persistence Implementation

4 Experimental Setup

4.1 Environment

Experiments were conducted on Google Colab using a T4 GPU (16GB VRAM) to support the inference of 7B parameter models in FP16 precision.

4.2 Models Evaluated

- **Qwen2-0.5B-Instruct:** A lightweight model used for sensitivity analysis.
- **Llama-2-7b-chat-hf:** A standard industry benchmark.
- **Mistral-7B-Instruct-v0.2:** A high-performance model known for coding capabilities.

5 Results and Analysis

To ensure a rigorous evaluation, we structured our analysis in two phases: a sensitivity analysis to verify metric stability, and a comparative analysis between models.

5.1 Phase 1: Individual Sensitivity Analysis

5.1.1 Qwen2-0.5B-Instruct

We tested Qwen with $N \in \{5, 50, 100\}$. As shown in Table 1, increasing N dramatically reduced the background noise (from 0.44 to 0.06). The Average MinHash stabilized around 0.06.

Table 1: Qwen-0.5B: Impact of Sample Size (N)

Metric	**N=5**	**N=50**	**N=100**
Avg MinHash Similarity	0.0515	0.0614	0.0602
Exact Match Rate	0.00%	0.00%	0.00%
Robustness Drop	0.0017	0.0051	0.0082
Background Noise (Baseline)	*0.4424*	*0.1172*	*0.0658*

5.1.2 Llama-2-7B-Chat

We repeated the process for Llama-2. The similarity score converged to ≈ 0.09 at $N = 100$. This slight increase over Qwen suggests Llama-2 captures better syntactic structure.

Table 2: Llama-2-7B: Impact of Sample Size (N)

Metric	**N=5**	**N=50**	**N=100**
Avg MinHash Similarity	0.0780	0.0943	0.0923
Exact Match Rate	0.00%	0.00%	0.00%
Robustness Drop	0.0258	0.0221	0.0103
Background Baseline	*0.4424*	*0.1172*	*0.0658*

5.1.3 Mistral-7B-Instruct

For Mistral, we extended the sensitivity analysis up to $N = 100$ to confirm the stability of the results. As shown in Table 3, Mistral consistently demonstrates the highest similarity scores among all models.

At $N = 100$, the Average MinHash Similarity stabilized at **0.1508**, which is significantly higher than the background noise baseline (0.0658). Despite this high structural similarity, the Exact Match rate remained at **0.00%**. The Robustness Drop slightly increased to 0.0161 but remains low enough to rule out overfitting.

Table 3: Mistral-7B: Impact of Sample Size (N)

Metric	**N=5**	**N=50**	**N=100**
Avg MinHash Similarity	0.1738	0.1475	0.1508
Exact Match Rate	0.00%	0.00%	0.00%
Robustness Drop	0.0010	0.0101	0.0161
Background Baseline	*0.4424*	*0.1172*	*0.0658*

5.2 Phase 2: Comparative Analysis

Table 4 compares the best-performing configurations for each model. Mistral-7B ($N = 100$) achieved the highest similarity (**0.1508**), indicating superior code generation quality. Crucially, it maintained a 0% Exact Match rate, proving it *understands* the coding patterns rather than *memorizing* specific examples.

Table 4: Final Cross-Model Comparison

Metric	**Qwen (0.5B)**	**Llama-2 (7B)**	**Mistral (7B)**
	(N=100)	*(N=100)*	*(N=100)*
Avg MinHash Similarity	0.0602	0.0923	**0.1508**
Exact Match Rate	0.00%	0.00%	0.00%
Robustness Drop	0.0082	0.0103	0.0161

6 Conclusion

We successfully implemented a modular framework to probe LLM memorization. Our extensive evaluation reveals that:

1. **Generalization over Memorization:** Across all tests, including adversarial settings, no model exhibited verbatim memorization (0% Exact Match).
2. **Model Quality:** Mistral-7B demonstrated the highest structural similarity (0.1508) to the ground truth, indicating superior understanding of coding patterns.
3. **Robustness:** All models showed negligible performance drops when docstrings were perturbed, confirming they rely on semantic understanding rather than specific triggers.

References

[1] P. Nguyen. “Projects and Exams for the Master Course Machine Learning for Software Engineering (DT1052).” University of L’Aquila, 2025.

[2] A. Z. Broder, “On the resemblance and containment of documents,” in *Compression and Complexity of Sequences*, 1997.