RUHR-UNIVERSITÄT BOCHUM

Memorization and Reasoning in Large Language Models

Michael Abraham van Veen, Olivier Christoph Wienchor, Florian Martin Kaiser
Forschendes Lernen im Modul Programmierung und Algorithmen
für ETIT und ITE (Technische Informatik 1)

Are LLMs reasoning — or just memorizing?

Motivation

Large Language Models sind als generative KI-Systeme in den vergangenen Jahren ihrer Funktion als Suchmaschine längst entwachsen und werden immer frequentierter in der Softwareentwicklung, Fehlerkorrektur und Codegenerierung eingesetzt. Aufgrund scheinbar individueller Reaktion auf wechselnde Problemstellungen wird ihnen häufig die Fähigkeit zugeschrieben, komplexe Probleme kausal zu durchdringen.

Eine zentrale Frage ist jedoch immer noch offen: Inwieweit basieren erzeugte Ausgaben auf echten Schlussfolgerungen, oder auf der Memorierung von Trainingsdaten, etwa begründet in Data Leakage?

Eben mit dieser Frage hat sich ein Forscherteam der Carnegie Mellon University (CMU, 2024) beschäftigt und in einer Veröffentlichung mit dem Titel **„Are Large Language Models Memorizing Bug Benchmarks“**[1] , dargestellt.

Kern der Untersuchungen waren hierbei zwei signifikante Indikatoren, **Negative Log Likelihood (NLL)** und **N-gram Accuracy**.

Basierend auf diesen zwei Metriken analysiert dieses Poster das Memorierungsverhalten großer Sprachmodelle und behandelt den Versuch unserer Forschungsgruppe an der Ruhr-Universität Bochum (2026) die Ergebnisse des Papers in einer reproduzierbaren Versuchsreihe zu verifizieren.

[1] Daniel Ramos u. a.: *Are Large Language Models Memorizing Bug Benchmarks?*, in: arXiv [Preprint], 31. März 2025, online unter: https://doi.org/10.48550/arXiv.2411.13323 (abgerufen am 28. Januar 2026).

Grundlagen und Vorgehen

Token: Kleinste Verarbeitungseinheit eines Large Language Models; Token entsprechen häufig Subwort-Einheiten und nicht zwingend ganzen Wörtern.

Membership: Beschreibt, ob ein Datenpunkt Teil der Trainingsdaten (im „Stack“) eines Modells war; wird meist über Membership-Inference-Methoden abgeschätzt.

Data Leakage: Überschneidung zwischen Trainings- und Evaluationsdaten, die zu verfälscht hoher Modellleistung durch Wiedererkennen statt Generalisierung führt.

Negative Log Likelihood (NLL): Maß für die Unwahrscheinlichkeit der vom Modell vorhergesagten Token; hohe Werte deuten auf hohe Überraschung hin, niedrige Werte auf Bekanntheit des Codes.

N-Gram Accuracy: Vergleicht vom Modell vervollständigte Token-Sequenzen mit einer Referenz und dient als Indikator für mögliche Memorierung. Hier: 5-Gram (5 Token)
In der Versuchsreihe des Teams der Carnegie Mellon University wurden programmiersprachenübergreifend Ground-Truth-Dateien aus sieben Bug-Benchmarks mit je zwischen 200 und 500 Bug-Fix Dateien analysiert.
Die Evaluation erfolgte auf sechs ausschließlich vortrainierten Base-Open-Source-LLMs ohne Instruction- oder Fine-Tuning, unter Verwendung der Metriken **Membership**, **Negative Log Likelihood (NLL)** und **N-Gram Accuracy**.

Zur Reduktion von Verzerrungen wurden sowohl im Versuchsaufbau der CMU, als auch in unserer Untersuchung von Duplikaten bereinigte Datensätze mit minimaler Übereinstimmung verwendet.

Für die NLL- und N-Gram-Berechnung wurden Dateien mittels überlappender Strides von 512 Token verarbeitet, um vollständigen Kontext sicherzustellen und die NLL-Werte aufgrund der Abhängigkeit von Tokenizer und Architektur nur innerhalb derselben Modellfamilie verglichen. Hierbei beschränkte sich unsere Versuchsreihe auf 30–50 Dateien pro Benchmark, um bei reduziertem Umfang vergleichbare und reproduzierbare Ergebnisse zu erzielen.

Allgemeiner Verfahrensablauf:

1.Auswahl und Download der Bug-Fix-Dateien
2.Download und Initialisierung der LLMs und Tokenizer
3.Tokenization
4.Striding
5.Messungen & Speichern der Ergebnisse in CSV-Datei
6.Auswertung

Abb. 1: GitLab-Repository mit verwendeten Skripts, Datei-referenzen, Ergebnissen und Original-paper-referenz

Aus unserer Versuchsreihe werden die Daten zu Mistral 7B, Codegen 6B Multi und Starcoder 7B aufgeführt. Die LLama-Modelle sowie Gemma2 27B sind anhand der beschriebenen Methodik lediglich auf die Ausgabe valider Daten geprüft worden, da die Größe der Modelle industrielle Hardware voraussetzt und die Rechenzeit aufgrund der VRAM Kapazität außerhalb eines umsetzbaren Rahmens lag.

Ergebnisse der Originalstudie

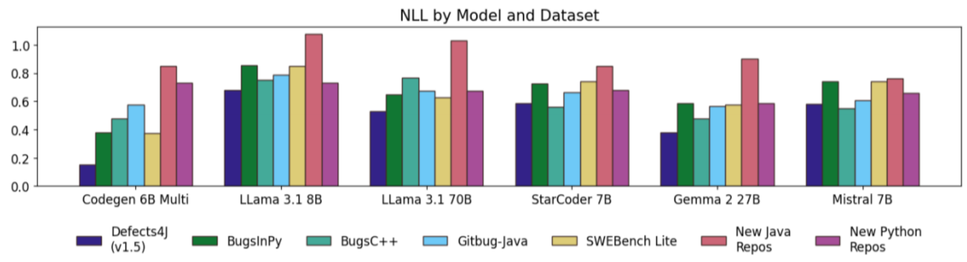


Abb. 2: Ergebnisse zur NLL der originalen Versuchsreihe

Die durchschnittlich niedrigsten NLL-Werte finden sich über allen Modellen bei Defects4J, was einen deutlichen Hinweis auf Data-Leakage bei diesem Benchmark ist. Insgesamt lässt sich erkennen, dass sich die NLL-Werte, bis auf eine Ausnahme von SWEBench Lite in Codegen 6B Multi, relativ konstant verhalten. Am unbekanntesten scheinen die Inhalte des New Java Repos zu sein, da hier die mit Abstand höchsten NLL-Werte zu finden sind. Auch lässt sich im Vergleich erkennen, dass die in Summe höchsten NLL-Werte bei der Llama-Modellfamilie auftreten.

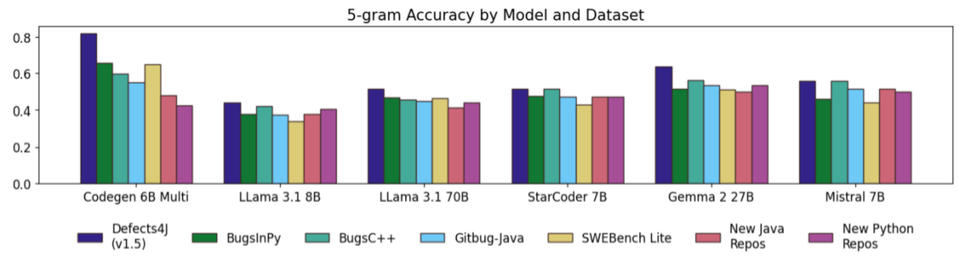


Abb. 3: Ergebnisse zur 5-gram Accuracy der originalen Versuchsreihe

Bei der 5-gram Accuracy lässt sich beobachten, dass unter allen Benchmarks Defects4J modellübergreifend die höchsten Werte erzielt. Alle anderen Benchmarks liefern vergleichsweise ausgewogene Ergebnisse innerhalb eines LLMs. Unter allen getesteten Modellen fällt Codegen 6B Multi ausnahmslos durch die höchsten Werte auf während die LLama-Modelle durchschnittlich am niedrigsten liegen.

Benchmark	v1.0 (%)	v2.0 (%)	v2.1 (%)
GitBug-Java	61.1	42.6	38.9
BugsInPy	94.1	64.7	64.7
BugsC++	60.9	60.9	65.2
Defects4J	80.0	80.0	80.0
SWEBench-Lite	83.3	91.7	83.3

Abb. 4 zeigt den prozentualen Anteil von Benchmarks an jeweiligen Stack Versionen.
Der „Stack“ bildet einen ausgewählten Trainingsdatensatz für LLMs.

1. Zur Bewertung potenzieller Data Leakage werden Membership, NLL und N-Gram Accuracy gemeinsam betrachtet, da keine einzelne Metrik isoliert aussagekräftig ist.
2. Ebenso impliziert Membership alleine keine Memorierung, sondern dient als Kontextinformation und wird gemeinsam mit NLL und N-Gram Accuracy interpretiert.
3. Modelle mit größerem Trainingsbudget zeigen insgesamt geringere Leakage-Signale, während größere Modellgrößen tendenziell höhere N-Gram Accuracy aufweisen.

Auswertung und Diskussion der Reproduktion

Insgesamt wurden für die reproduzierte Statistik 291 Dateien mit 39.945 5-gram-Accuracy Berechnungen und 57.161.090 NLL-Tokens ausgewertet.

Codegen 6B Multi zeigte in unserer Evaluierung(Abb. 5-6), aufgrund der größten Abdeckung, die höchste Übereinstimmung mit den Ergebnissen des Originalpapers.

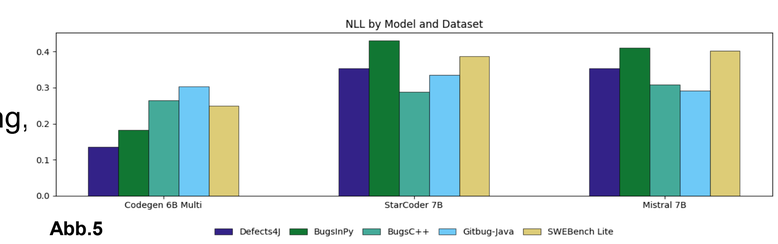


Abb.5

StarCoder 7B und Mistral 7B zeigen ähnliche relative Tendenzen über die Benchmarks hinweg (insbesondere BugsInPy, BugsC++ und Gitbug-Java), weichen in absoluten Werten jedoch weiter von den Werten des behandelten Papers ab, als es bei Codegen 6B Multi der Fall ist.
Auffällig sind die vergleichsweise niedrigen Defects4J-Werte bei diesen Modellen.

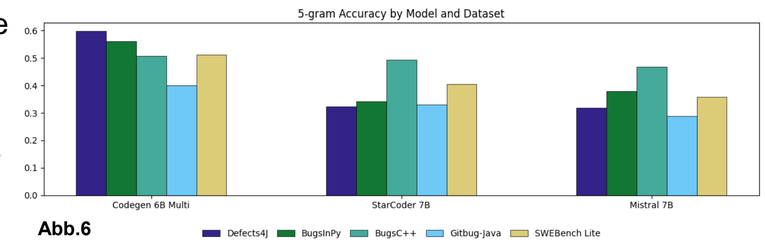


Abb.6

Trotz Abweichungen konsistente Ergebnisse innerhalb der Versuchsreihen lassen den Schluss auf eine valide Implementierung der Methodik zu und bestätigen die Ergebnisse des behandelten Papers. Unterschiede in den absoluten Werten lassen sich plausibel durch geringere Benchmark-Abdeckung, Dateiaktualisierungen, Modellversionen sowie die Implementierung des Verfahrens erklären.
Insgesamt konnte eine reproduzierbare Versuchsreihe aufgebaut und zur Verfügung gestellt werden, die die Kernaussage des Papers der Carnegie Mellon University bestätigt.