LLMs and Prompting for Unit Test Generation: A Large-Scale Evaluation

Wendkûuni C. Ouédraogo, Kader Kaboré, Yewei Song, Jacques Klein
firstname.lastname@uni.lu
University of Luxembourg

Haoye Tian
firstname.lastname@uni.lu
University of Melbourne

Anil Koyuncu
firstname.lastname@uni.lu
Bilkent University

David Lo
davidlo@smu.edu.sg
Singapore Management University

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu
University of Luxembourg

ABSTRACT

Unit testing, essential for identifying bugs, is often neglected due to time constraints. Automated test generation tools exist but typically lack readability and require developer intervention. Large Language Models (LLMs) like GPT and Mistral show potential in test generation, but their effectiveness remains unclear.

This study evaluates four LLMs and five prompt engineering techniques, analyzing 216 300 tests for 690 Java classes from diverse datasets. We assess correctness, readability, coverage, and bug detection, comparing LLM-generated tests to EvoSuite. While LLMs show promise, improvements in correctness are needed. The study highlights both the strengths and limitations of LLMs, offering insights for future research.

KEYWORDS

Automatic Test Generation, Unit Tests, Large Language Models, Prompt Engineering, Empirical Evaluation

ACM Reference Format:
Wendkûuni C. Ouédraogo, Kader Kaboré, Yewei Song, Jacques Klein, Haoye Tian, Anil Koyuncu, David Lo, and Tegawendé F. Bissyandé. 2024. LLMs and Prompting for Unit Test Generation: A Large-Scale Evaluation. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3691620.3695330

1 INTRODUCTION

Unit testing ensures code functions correctly, catching bugs early and maintaining quality. However, manual test creation is time-consuming and often neglected. Automated tools like EvoSuite assist but produce tests that lack readability and need manual intervention, reducing their effectiveness.

Large Language Models (LLMs) like GPT and Mistral have shown promise in automating test generation, producing correct and readable tests. However, questions remain about their true effectiveness in generating comprehensive tests that detect real-world bugs and how different prompting techniques influence their performance.

The main goal of this research is to evaluate how effective LLMs are at generating unit tests suite, focusing on how prompt engineering techniques impact performance. The study assesses LLM-generated tests in terms of correctness, readability, coverage, and bug detection, comparing them with traditional tools like EvoSuite.

This study undertakes a large-scale evaluation of four advanced LLMs (GPT-3.5 [12], GPT-4 [1], Mistral 7B [8], and Mixtral 8x7B [9] and five distinct prompt engineering techniques namely Zero-shot learning (ZSL), Few-shot learning (FSL), Chain-of-Thought(CoT), Tree-of-Thoughts (ToT), and Guided Tree-of-Thoughts (GToT) a hybrid approach that combines the structured exploration of ToT with additional guidance from CoT to focus on relevant reasoning paths. By analyzing 216 300 tests generated for 690 Java classes from diverse datasets, this research provides the first comprehensive comparison between LLM-generated tests and those produced by traditional tools like EvoSuite. The findings reveal both the strengths and limitations of LLMs in this domain, offering critical insights into their potential to transform software testing practices.

2 OVERVIEW OF APPROACH

2.1 Usage Scenario

In a Java project, a developer uses an LLM to automate unit test creation for a key class. By providing the class's source code and a prompt, the LLM generates a suite of tests covering all methods, including edge cases, enhancing software reliability. For instance, the developer can prompt the LLM to generate a comprehensive JUnit 4 test file for the class [1].

2.2 Technical Overview

In this study (Figure 1), we evaluated four prominent LLMs: GPT-3.5, GPT-4, Mistral 7B, and Mixtral 8x7B—using prompt engineering techniques like Zero-shot learning [16], Few-shot [15], CoT [18], ToT [19], and GToT. The evaluation utilized three datasets: SF110 [6], a benchmark with 110 Java projects; Defects4J [10], containing real-world bugs from open-source projects; and a Custom Dataset with Java classes and bugs from recent projects. This dataset focused on well-maintained projects from the OceanBase Developer Center (ODC) [11] and Conductor OSS [4].

[1] https://chatgpt.com/share/99d6468e-98ed-40b1-9b14-17bb9c239f9c

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10.
https://doi.org/10.1145/3691620.3695330

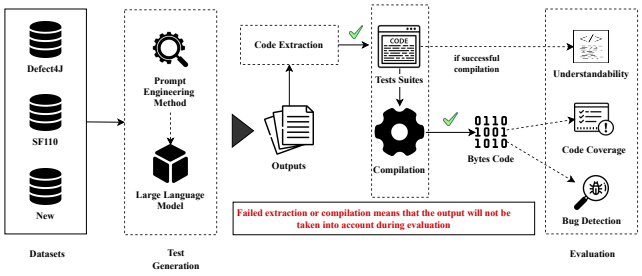

Figure 1: Overview of the pipeline used to design the experiments.

3 EXPERIMENTAL SETUP AND RESULTS

3.1 Experimental Setup

To address the research questions, we conducted evaluations using three datasets: SF110, Defects4J, and a Custom Mini Dataset (CMD). We utilized several tools, including EvoSuite [5] with its DynaMOSA [13] algorithm for maximizing coverage, SpotBugs [17] for detecting bugs and suboptimal coding practices, JaCoCo [7] for measuring code coverage, and Checkstyle [3] for ensuring adherence to coding standards (Sun Code Conventions and Google Java Style). PMD [14] was used to assess the cognitive and cyclomatic complexity of the test suites, while Javalang [2] verified syntactical correctness. Each experiment was repeated 30 times to account for randomness in LLM outputs and EvoSuite's algorithms.

3.2 Experimental Results

RQ1: How does prompt engineering influence the correctness and compilability of generated test suites? The study found that prompt engineering greatly affects the correctness and compilability of LLM-generated test suites. Guided Tree-of-Thoughts (GToT) had the highest rate of correct and compilable tests, while Zero-shot Learning (ZSL) performed lower. Structured prompts consistently improved test correctness across all LLMs.

RQ2: How do LLM-generated test suites compare to those produced by EvoSuite in terms of code coverage? EvoSuite typically achieved higher line and method coverage, but LLMs using Chain-of-Thought (CoT) and GToT approaches matched or exceeded EvoSuite's coverage in some cases, especially on the Custom Mini Dataset. While EvoSuite remains a strong baseline, LLMs show promise in generating effective test suites with proper prompts.

RQ3: What is the effectiveness of LLM-generated tests in detecting bugs? LLM-generated tests varied in bug detection across datasets and prompts. On Defects4J, CoT and GToT prompts detected many bugs, occasionally surpassing EvoSuite. However, detection rates were lower on the CMD, suggesting LLMs struggle with more complex or unfamiliar code. Nevertheless, with proper prompts, LLMs can complement tools like EvoSuite in bug detection.

4 CONCLUSION

Our study shows that prompt design significantly impacts LLM-generated unit tests, with Chain-of-Thought and Guided Tree-of-Thought prompts yielding the best results. While LLMs face correctness challenges, they produce readable, human-like code and perform well in bug detection, especially on familiar datasets. Though EvoSuite generally achieves higher coverage, LLMs, when well-prompted, can complement traditional tools. Further research is needed to improve LLM performance on complex or unfamiliar codebases.

5 ACKNOWLEDGEMENT

This work is supported by funding from the Fonds National de la Recherche Luxembourg (FNR) under the Aides à la Formation-Recherche (AFR) (grant agreement No. 17185670).

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
[2] c2nes. 2012. javalang. https://github.com/c2nes/javalang. [Online; accessed 06-Jun-2024].
[3] checkstyle. 2001. Checkstyle. https://checkstyle.sourceforge.io/. [Online; accessed 06-Jun-2024].
[4] conductor oss. 2023. Conductor OSS). https://github.com/conductor-oss/conductor. [Online; accessed 06-Jun-2024].
[5] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.
[6] Gordon Fraser and Andrea Arcuri. 2014. A large-scale evaluation of automated unit test generation using evosuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 1–42.
[7] JaCoCo. 2022. JaCoCo Java Code Coverage Library. https://www.jacoco.org/jacoco/. [Online; accessed 06-Jun-2024].
[8] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
[9] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).
[10] René Just, Darioush Jalali, and Michael D Ernst. 2014. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*. 437–440.
[11] OceanBase. 2023. OceanBase Developer Center (ODC). https://github.com/oceanbase/odc. [Online; accessed 06-Jun-2024].
[12] OpenAI. 2023. GPT-3.5-Turbo. *URL: https://platform.openai.com/docs/models/gpt-3-5-turbo/[accessed 2024-05-21]* (2023).
[13] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.
[14] PMD. 2012. PMD An extensible cross-language static code analyzer. https://pmd.github.io/. [Online; accessed 06-Jun-2024].
[15] Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–7.
[16] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv preprint arXiv:2402.07927* (2024).
[17] spotbugs. 2016. Spotbugs. https://spotbugs.github.io/index.html. [Online; accessed 06-Jun-2024].
[18] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
[19] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).