# Quality Assessment of LLM Models Generated Unit Tests: Quality Metrics Completeness from Code-aware Perspective

Dovydas Marius Zapkus, Asta Slotkienė

## Introduction

Unit testing is a fundamental aspect of software testing, which ensures the correctness and robustness of code implementations, but their creation requires considerable time and resources from developers. It has already been proven that special software can generate test cases using conventional methods such as search-based software testing or random testing (Tang et al., 2024). However, the generated test's code reached high code coverage metrics but was highly unreadable by developers. Large Language Models (LLMs) can solve readability issues by learning from training data containing real human-written test code examples. However, another challenge arises, such as unit tests reaching better coverage, but they are independent of functional context (Ryan et al., 2024). Researchers suggest solving this issue by additionally introducing the context of the code fragment into LLM's training set, improving overall results and its quality metrics. Recent research with LLM-generated unit tests focuses on code coverage as a unit test quality measure (Pan et al., 2024; Lops et al., 2024; Bhatia et al., 2024). However, this is not enough, and we suggest involving additional ways in which unit tests will be reliable and understandable. These additional two ways: comparison measures based on abstract syntax trees such as CodeBLEU and RUBY. According to this, this paper proposed to research and analyze how to measure the quality of the LLM-generated unit tests. In this research, three LLM models were applied, which were used for unit test generation according to the provided source codes. The generated unit tests were evaluated by test quality metrics such as coverage and machine translation-based metrics. Our research results allow us to highlight several results of generated unit tests with several LLM models: Gemini Pro, Claude 3.5, and GPT 4.0.

The first observation was that LLM models generated unit test coverage that achieved from 34% to 100%, and mutation tests covered from 27-73%. The second research result was that semantic and syntactic similarity based on AST - was achieved from 0,35 until 0,83 when the unit tests were generated from source code with low cyclomatic code complexity and loose class coupling. When the source code is more complex, LLMs could achieve from 0,28 to 0,8 semantic and syntax similarity.

## Research Methodology

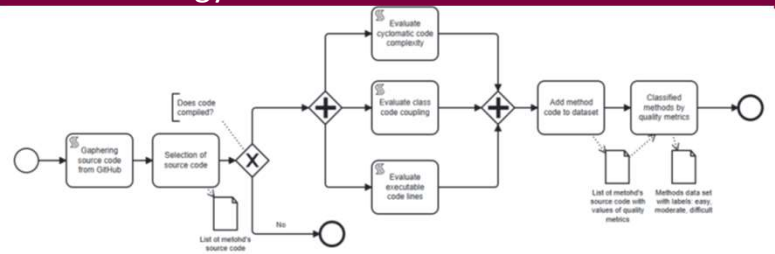


**Figure 1**. Principal scheme of method's source code gathering process

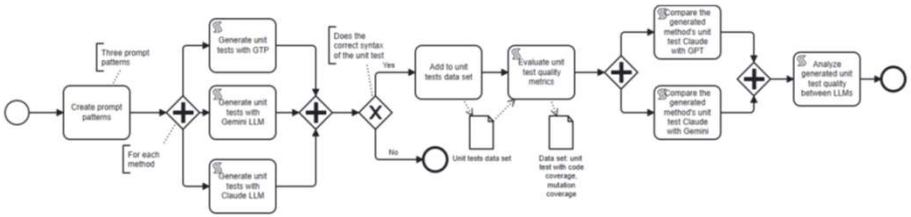


**Figure 2**. Principal scheme of research for unit code quality evaluation

Abstract syntax tree (AST)measures include CodeBLEU measurement, which is a measurement derived from the BLEU quality metric. BLEU considers n-gram match but ignores code structure, which is crucial for code evaluation. CodeBLEU proposes weighted n-gram match and syntactic abstract syntax tree match to measure grammatical correctness and introduce semantic data-flow match to calculate logic correctness. CodeBLEU is calculated with the formula below (Ren et al. 2020):

$$CodeBLEU = \alpha \cdot BLEU + \beta \cdot BLEU_{weight} + \gamma \cdot Match_{ast} + \delta \cdot Match_{df}$$

Another metric that measures abstract syntax tree is RUBY. The RUBY metric was suggested by Tran et al. (2019) as an alternative to the natural language metrics. The metric compares program dependency graphs (PDG) of the reference and the candidate; if a PDG is impossible to build, it falls back to comparing AST, and if an AST is also impossible to build, the metric compares the weighted string edit distance between the (tokenized) reference code Rc and the generate code Gc sequences.
The RUBY metric is defined as:

$$RUBY\ (Rc, Cc) = \begin{cases} GRS\ (Rc, Gc), & if\ PDGs\ are\ applicable \\ TRS(Rc, Gc), & if\ ASTs\ are\ applicable \\ STS\ (Rc, Gc), & otherwse \end{cases}$$

*GRS - graph similarity, TRS – AST tree similarity; STS - string similarity;*

## Results

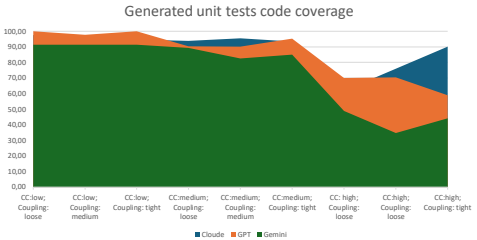


**Figure 3**. Generated unit tests quality evaluation: code coverage metric

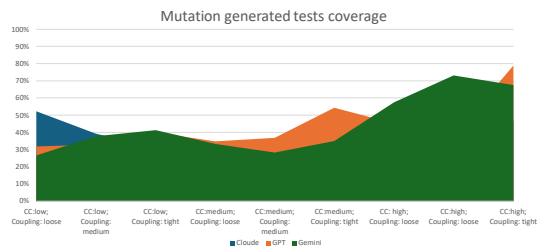


**Figure 4**. Mutation coverage of generated unit tests

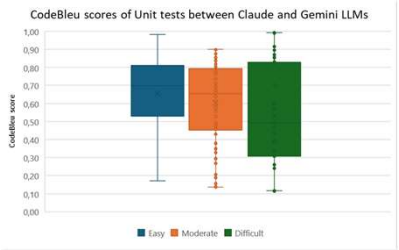


**Figure 5**. Comparison of CodeBleu scores

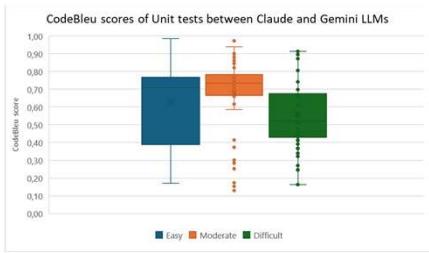


**Figure 5**. Comparison of CodeBleu scores

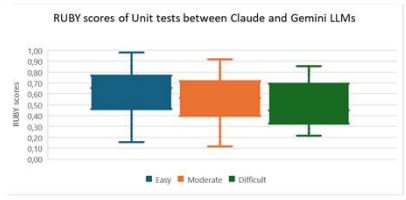


**Figure 6**. Comparison of RUBY scores

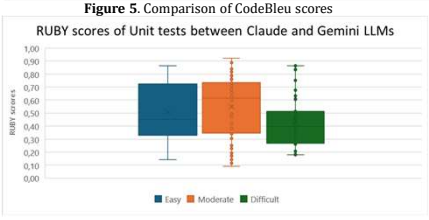


**Figure 7**. Comparison of RUBY scores

## References

Tang, Y., Liu, Z., Zhou, Z., & Luo, X. (2024). Chatgpt vs SBST: A comparative assessment of unit test suite generation. IEEE Transactions on Software Engineering.
Ryan, G., Jain, S., Shang, M., Wang, S., Ma, X., Ramanathan, M. K., & Ray, B. (2024). Code-Aware Prompting: A Study of Coverage-Guided Test Generation in Regression Setting using LLM. Proceedings of the ACM on Software Engineering, 1(FSE), 951-971.
Pan, R., Kim, M., Krishna, R., Pavuluri, R., & Sinha, S. (2024). Multi-language Unit Test Generation using LLMs. arXiv preprint arXiv:2409.03093.
Bhatia, S., Gandhi, T., Kumar, D., & Jalote, P. (2024, April). Unit test generation using generative AI: A comparative performance analysis of autogeneration tools. In Proceedings of the 1st International Workshop on Large Language Models for Code (pp. 54-61).
Lops, A., Narducci, F., Ragone, A., Trizio, M., & Bartolini, C. (2024). A System for Automated Unit Test Generation Using Large Language Models and Assessment of Generated Test Suites. arXiv preprint arXiv:2408.07846.
Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., ... & Ma, S. (2020). Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.
Tran, N., Tran, H., Nguyen, S., Nguyen, H., & Nguyen, T. (2019, May). Does BLEU score work for code migration?. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)* (pp. 165-176). IEEE.
Kvedaravičius Tomas (2024). Bachelor thesis. The Quality Assessment of Unit Test Generation for Large Language Models. 2024, Vilniaus University (supervisor Asta Slotkienė)