Improving Coverage Based Fault Localization through LLM-based Test Generation

Ashrit Reddy Yarava
Electrical and Computer Engineering
University of Alberta
Edmonton, Alberta
yarava@ualberta.ca

***Abstract*—Spectrum-based fault localization (SBFL) is one popular coverage based FL algorithm. It uses the test coverage to identify the most suspicious code elements. Other FL approaches (LLM, DL) also make use of the test suite to determine the faulty code elements. As a result, the quality of a test suite plays a major role in FL performance. Tests are usually written to maximize coverage and ignore their impact on FL. The proposed algorithm uses LLMs to introduce new unit tests to improve FL.**

***Index Terms*—test generation, llm, fault localization**

I. Background

This section provides an overview of the relevant research and concepts related to the proposal.

A. Fault Localization

Fault localization (FL) aims to find the faulty elements in a codebase. There are a variety of paradigms for FL, ranging from LLMs, Mutations, and Deep Learning. One shared trait between these algorithms is that they rely on the test suite to some extent. As such, improvements to the test suite are likely to result in FL improvement.

1) LLM Based Fault Localization: LLMs are great for analyzing code. LLM based FL approaches identify behavior in failing tests and code to find faults. [1], [2] provides the LLM with tools to navigate the codebase. [3] uses summarization to narrow down the fault from a class-level to method-level. These approaches make use of the failing tests. Generating more failing tests would add more information to the LLM.

2) Learning Based Fault Localization: Deep learning is also used for FL. In [4], the authors make use of a trained graph neural network to learn connections between methods. This approach makes use of the passing and failing tests. The proposed test generation algorithm would likely improve performance.

3) Mutation Based Fault Localization: This approach makes small changes to the codebase to identify faulty code. Mutations made to faulty code should not result in changes to test results. [5] is a popular MBFL tool that makes use of this philosophy. Improvements to the tests would add more points of failure preventing incorrect answers.

4) Spectrum Based Fault Localization: SBFL is a coverage based FL algorithm that relies on the test coverage of passing and failing tests. Ranking functions such as Ochiai make use of these counts to assign suspiciousness. It works at different levels of granularity such as class, method or line level.

A variety of algorithms make use of SBFL as the basis for tasks such as automated program repair (APR). In fact, [6] finds that almost all APR tools make use of SBFL in the FL step. These approaches use SBFL due to the efficiency and speed that the algorithm offers. The paper also identifies that limited SBFL performance hinders APR. As such, improving SBFL is a key research field and has implications for a variety of other fields.

B. Test Generation for FL

[7] introduced the idea of basic blocks, and their detrimental impact on FL. Basic blocks are sets of elements that have the same test coverage. In SBFL, having the same test coverage means having the same suspiciousness score. Large basic blocks can result in ties where a set of elements have equal suspiciousness. [7] proposes that tests generated with FL in mind should aim to maximize basic blocks in the codebase. The authors tackle the problem of test generation for FL with genetic algorithms. While useful, genetic algorithms (GA) are unable to generate complex tests.

Another explored approach [8] breaks down existing test cases to smaller subcomponents. Through this breakdown, the test cases are ”purified.” The increase in test cases also splits the execution paths increasing basic blocks.

These existing approaches only test performance improvement in SBFL. They do not identify how test generation would impact other FL algorithms.

C. LLM-Based Test Generation

The advent of powerful LLMs such as ChatGPT and DeepSeek have seen their use in a variety of SWE tasks. [9]–[11] show that LLMs can generate complex tests and outperform classical approaches. These approaches focus on maximizing text coverage and ignore FL performance. These approaches also do not generate tests with a specific intent. [12] tackles test generation with a given intent like the proposed approach.

D. Primary Contribution

While there is active research in test generation and FL, there is a gap on how to improve FL with test generation.

The proposed algorithm evaluates how test generation can impact FL algorithms. Prior studies on the combination of test generation and FL only study SBFL. The proposed approach broadens the scope to other FL paradigms.

II. Planned Approach

The following section provides an overview of the algorithm, experimental design, and potential challenges the algorithm faces.

A. Overview of the Algorithm

The algorithm consists of three components. The first component, bug report generation, summarizes the failing tests and stack traces. Basic block identification and selection is the next step. It involves constructing basic blocks using the test coverage. The Ochiai function determines the suspiciousness scores for each basic block. The most suspicious basic block is then selected for test generation. The third step, test generation, generates a unit test for each method in the basic block. Failing tests are more valuable for FL than passing tests. As such, the step generates tests to search for a possible failing tests. The loop stops when a condition passes to ensure the algorithm does not waste energy. The LLM is also given the bug report to encourage generating fault-relevant tests. The algorithm is then evaluated on a variety of FL approaches.

B. Experimental Design

We test the algorithm on the Defects4J v3.0 [13] dataset, a collection of java projects. For each project, we record the original FL performance. We run the algorithm and record the changed FL performance to track improvement. We use the SBFL, AutoFL, Metallaxis, and GRACE algorithms for evaluation of test generation. We test FL performance on the top@k and MFR metrics. Top@k is the first k most suspicious methods. MFR is the mean ranking of the highest ranked faulty method. We do not include the MAR metric since the algorithm only optimizes the first basic block. As such, it is likely that the first basic block might not contain all faulty methods. Finally, there will be an ablation study to identify the impact of components.

C. Tools and Techniques

We write the algorithm in Java and explore a variety of LLMs for the test generation step. We use SootUp and JavaParser for parsing the java code and use static analysis for test coverage.

D. Potential Challenges

There are a list of possible approaches for the stop condition in test generation. Each of these approaches should explored. The first approach involves comparing execution paths between existing and new tests. The second approach involves comparing the information in the prior and new test.

Test generation with LLMs encounters errors in generation, usually leading to compilation failures. As seen in [14], LLMs are prone to minor compilation errors. We use a basic iterative approach to fix compilation errors but it is possible to run into errors. This approach might not be enough to have a large enough success rate.

III. Project Timeline

A rudimentary version of the algorithm is currently functional. The next step in implementation builds the fault-triggering test search loop. Different stop conditions for the search loop also need to be explored.

Testing will begin with SBFL due to the current poor performance compared to other FL algorithms. Later on, different approaches will be used based on code available online.

References

[1] S. Kang, G. An, and S. Yoo, "A quantitative and qualitative evaluation of llm-based explainable fault localization," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1424–1446, 2024.

[2] H. Cho, S. Kang, G. An, and S. Yoo, "Cosmosfl: Ensemble of small language models for fault localisation," in *2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, pp. 17–24, IEEE, 2025.

[3] Y. Qin, S. Wang, Y. Lou, J. Dong, K. Wang, X. Li, and X. Mao, "S oap fl: A standard operating procedure for llm-based method-level fault localization," *IEEE Transactions on Software Engineering*, 2025.

[4] Y. Lou, Q. Zhu, J. Dong, X. Li, Z. Sun, D. Hao, L. Zhang, and L. Zhang, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pp. 664–676, 2021.

[5] M. Papadakis and Y. Le Traon, "Metallaxis-fl: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.

[6] K. Liu, A. Koyuncu, T. F. Bissyandé, D. Kim, J. Klein, and Y. Le Traon, "You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems," in *2019 12th IEEE conference on software testing, validation and verification (ICST)*, pp. 102–113, IEEE, 2019.

[7] B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," in *Proceedings of the 28th international conference on Software engineering*, pp. 82–91, 2006.

[8] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pp. 52–63, 2014.

[9] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "Chatunitest: A framework for llm-based test generation," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 572–576, 2024.

[10] Z. Wang, K. Liu, G. Li, and Z. Jin, "Hits: High-coverage llm-based unit test generation via method slicing," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1258–1268, 2024.

[11] Z. Zhang, X. Liu, Y. Lin, X. Gao, H. Sun, and Y. Yuan, "Llm-based unit test generation via property retrieval. arxiv 2024," *arXiv preprint arXiv:2410.13542*.

[12] Z. Nan, Z. Guo, K. Liu, and X. Xia, "Test intention guided llm-based unit test generation," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 779–779, IEEE Computer Society, 2025.

[13] R. Just, D. Jalali, and M. D. Ernst, "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*, ISSTA 2014, (New York, NY, USA), p. 437440, Association for Computing Machinery, 2014.

[14] M. Konstantinou, R. Degiovanni, J. M. Zhang, M. Harman, and M. Papadakis, "Yate: The role of test repair in llm-based unit test generation," *arXiv preprint arXiv:2507.18316*, 2025.