evoLLve'M: Improving JUnit Test Assertions and Mutation Score Using ChatGPT-4o and EvoSuite

Arda Turhan[1]
Supervisor(s): Annibale Panichella[1]**, Mitchell Olsthoorn**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Arda Turhan
Final project course: CSE3000 Research Project
Thesis committee: Annibale Panichella, Mitchell Olsthoorn, Casper Poulsen

evoLLve'M: Improving JUnit Test Assertions and Mutation Score Using ChatGPT-4o and EvoSuite

Arda Turhan
Delft University of Technology
Delft, The Netherlands

Mitchell Olsthoorn
Delft University of Technology
Delft, The Netherlands

Annibale Panichella
Delft University of Technology
Delft, The Netherlands

ABSTRACT

Software testing is a vital yet time consuming process during the development lifecycle, often causing engineers to limit its use in practice. In order to encourage active software testing, researchers have shown significant advances in automatic unit test case generation with approaches such as search-based testing (i.e., EvoSuite) and large language models (i.e., ChatGPT). However, while the first suffers with exploring edge cases of the input space, the latter still suffers from hallucinations during code synthesis, limiting the use of both solutions. This research aims to overcome these limitations by utilizing the strengths of both techniques, which are effective test structure generation and program inference, respectively. In particular, the assertions of initial unit tests generated by EvoSuite are augmented using ChatGPT-4o, with the aim of improving the mutation score, and hence the overall test suite effectiveness. We evaluate our solution, called evoLLve'M, on a benchmark of 20 Java classes from the SourceForge110 Corpus and compare it to only using EvoSuite, which is considered the state-of-the-art approach. Results show that evoLLve'M outperforms EvoSuite in 25% of the classes for mutation score, without negatively impacting other classes. It boosts the total number of killed mutations by 3%, achieving the most improvement for mutations types of increments and null returns, being 26.9% and 8.9%, respectively.

1 INTRODUCTION

Software testing minimizes critical system failures, which can range from bugs causing direct financial loss, to usability issues undermining end-user trust [27]. Though being expensive and time-consuming to perform, it comes to no surprise that testing urges developers to maintain a high quality of the software itself [13]. In particular, unit testing forms the foundation of this process [22]. A unit test consists of two primary components: a test-prefix, which is a combination of input and program statements that produce the desired state to be tested, and a test oracle, which verifies the expected result of this state with the actual output. Prior research shows that oracles can be effectively implemented by using test assertions [2]. Hence, using assertions we can create test cases that aim to distinguish correct system behavior from incorrect behavior, providing a necessary baseline for the quality of software.

Nevertheless, existing testing practices do not sufficiently focus on unit testing nor on assertions [3, 7], as they are time-consuming to write and developers do not know which assertions to make. This is not unexpected, as the objective to distinguish correct from incorrect program behavior is an undecidable problem [2]. However, we can still create robust tests suites that ensure high effectiveness by aiming to maximize the confidence of this distinction. Literature indicates that this is best measured by mutation score [11, 21, 29], and is strongly correlated with test assertions [30]. Therefore, we hypothesise that by improving the test assertions, we will also increase the mutation score, implying a higher effectiveness of the overall test suite.

Recent works shows large advances using approaches that generate tests using LLMs. This includes work where multiple models are first fine-tuned on additional data [8]. Other works shows the impact of prompt engineering, where iteratively prompting the LLM with findings from intermediate mutation testing lead to increased mutation scores [5]. However, a common deficit is that fine-tuning is an expensive operation, requiring vast amounts of data. Furthermore, most solutions either evaluate on publicly known datasets which could result into bias, as well as reporting on code coverage rather than mutation score.

In this paper, we address these limitations by using the strengths of search-based test generation from EvoSuite, together with those of OpenAI's flagship large language model (LLM), ChatGPT-4o. More specifically, we make use of effective test structure generation and program inference, respectively. EvoSuite's state-of-the-art evolutionary algorithm, DynaMosa [19], will generate an initial JUnit test for each evaluated Java class. Subsequently, the LLM will improve assertions by augmenting these tests through few-shot prompting, where we aim to minimize the effect of code hallucinations since tests do not have to be generated from scratch [6].

In order to evaluate the performance of the proposed solution, that is, evoLLve'M, we performed an empirical study that investigates its impact on: i) mutation score, and ii) type of mutations killed. We compare these results with only using Evosuite. In particular, we will conduct these experiments with 20 classes obtained from the SourceForge110 Corpus as a benchmark. These projects are not (yet) listed on well known developer platforms such as GitHub nor used frequently in research. Hence with high likelihood, are not included in the training set for ChatGPT-4o, which could skew the outcome of the study [12].

The results show that augmenting search-based generated tests by large language models leads to a higher mutation score for 25% of the evaluated classes, while not negatively impacting the others. Furthermore, while showing similar results in killed mutations per type, it achieves a 26.9% and 8.9% increase in killed mutations for the mutation types increment and null returns, respectively.

The remainder of this paper is structured as follows: Section 2 provides a background on two widely-used methods in research to generate unit tests, namely search-based generation and that with LLMs. Following this, Section 3 introduces a solution where both methods are utilized to create more effective test assertions. Section 4 describes which experiments will be used for evaluation to assess the improvement, of which the results and analysis will be presented in Section 5. Threats to validity of the observed findings shall be addressed in Section 6. Section 7 is dedicated to responsible research, reflecting on the reproducibility of the experiments and its ethical and societal impact. Finally, Section 8 concludes the paper with some thoughts on future directions.

2 BACKGROUND AND RELATED WORK

In this section, we introduce the background of two widely-used methods in research to automate unit test-case generation, namely search-based test generation and test generation by using LLMs.

2.1 Search-Based Test Generation

Search-based software testing is a technique that approaches the objective of generating unit tests as an optimization problem. Here, the main idea is to generate tests by leveraging evolutionary algorithms that navigate (i.e., search) the input space of the program based on meta-heuristics, while maximizing for one or multiple search criteria. Various approaches have been proposed in prior literature, maximizing for criteria such as statement and branch coverage, and achieving significant results while making use of genetic algorithms [4, 16].

In particular, EvoSuite has been proven to be the state-of-the-art search-based unit test generation tool for Java, repeatedly beating other existing approaches at the annual Search-Based and Fuzzing Testing (SBFT) Tool competition [10], as well as outperforming approaches using ChatGPT [26]. By default, it uses the DynaMosa [19] algorithm which is proven to be superior for testing compared to other available evolutionary algorithms [20]. This study uses EvoSuite to generate an initial unit test for subsequent improvement, which can lead to satisfactory results as introduced by [18]. For comparison, we will later use these initial tests as the baseline for evaluation. Specific configurations for the use of EvoSuite in this research will be discussed in section 4.2.

2.2 LLM Based Test Generation

Recenlty, test generation using LLMs has initiated a large momentum in research interest, having shown success in similar tasks such as question-answering and text generation []. Here, the use of LLMs for test generation in prominent work is typically assisted by either prompt engineering [27] or fine-tuning the LLM with an additional corpus of data, as large generative code models are found to not be well-calibrated for this task straight out of the box [15, 24].

Recent work showing satisfactory results include, but are not limited to, TOGLL [8], where multiple models are fine-tuned on the SF110 Corpus for better results in test oracle generation. Rather than fine-tuning, MUTAP [5] shows high mutation score by iteratively prompting the LLM with findings from intermediate mutation testing. Combining both search-based testing as LLMs, CODAMOSA [14] aims to escape the coverage plateau of search-based approaches intermittently injecting improved test inputs generated by an LLM, during the search of the evolutionary algorithm in use.

However, a common deficit all solutions share is that generally, LLMs do not perform well enough yet when generating tests from scratch for programs outside of their training set [6, 25]. This was also reflected in work where test generation capabilities of multiple widely-used LLMs were exercised on well-known datasets such as HumanEval, available on GitHub, with that of the lesser-known SF110 Corpus, highlighting resulting code coverage of the first by 80%, while only reaching 2% for the latter [23]. These observations necessitate further research on using LLMs not to generate tests from scratch, but to combine its unique inference abilities with other existing methods for test generation, reducing LLM code hallucinations [6]. Last, it is important to note that most work only reports on code coverage metrics (not including assertion coverage), which has been proven to be an inadequate measure for test suite effectiveness [9].

3 APPROACH

This section presents our approach, called evoLLve'M, which aims to generate unit tests with meaningful test assertions by utilizing the strengths of both search-based test generation and that of LLMs. As shown in Figure 1, the approach uses EvoSuite to create an initial JUnit test case for each Program Under Test (PUT), of which the assertions will be further improved by ChatGPT-4o using few-shot prompting.

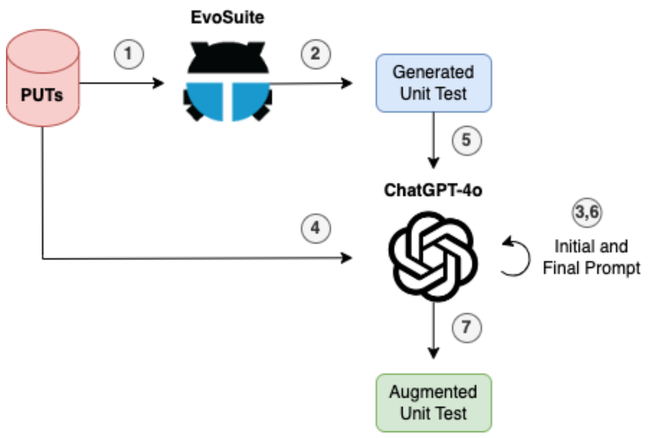

Figure 1: The proposed approach for generating JUnit tests with meaningful assertions using EvoSuite and ChatGPT-4o.

3.1 Initialization

In order to create initial unit tests for each PUT, we use EvoSuite with the DynaMosa algorithm. We motivate this decision as it is the state-of-the-art technique for JUnit test generation, also generating structured unit test cases as mentioned prior. However, some initial unit tests may contain EvoSuite specific imports or method calls, such as those used for mocking or exception handling. While appearing infrequently, this can cause tests to not compile when used outside of the EvoSuite environment. These statements are manually removed to mitigate this effect, also halting their propagation to subsequent stages of the approach. Furthermore, the resulting unit tests will not only be used for prompting, but also form the baseline for our evaluation.

3.2 Assertion Augmentation

To improve assertions of the initialized unit tests for each PUT, the following process of prompting is divided into four steps. We will first describe each step, after which we give an overview of a full prompt for a PUT. All prompting is done with API calls using the official OpenAI Python library. The first step of the augmentation process is indicated by ③ in Figure 1. We create a new ChatGPT-4o session, prompting it to consider a java class and corresponding unit test, sent subsequently over separate prompts. We also include it should not give a respond yet, and wait for further instructions after having received both inputs. In the following two steps, we provide both the PUT and corresponding unit tests as strings, indicated by ④ and ⑤ respectively. Again, we motivate the decision to provide an effective and structured unit test during this process, as literature shows correct examples of code can reduce hallucinations of LLMs [6], also specifically for assertion related tasks [26]. Finally, ⑥ represents the last prompt, where we prompt to improve the assertions of the given unit test such that it will result into a higher mutation score. Moreover, the final prompt includes that we do not want any further explanation, resulting in only the code of the augmented unit test being returned. For the outgoing prompt in each step, we include previous prompts and responses for the current session, if applicable. A full overview of prompting for a PUT is as following:

1) "Please consider the following java class, and also a corresponding unit test for it. I shall upload them in subsequent prompts, first the class and then the test. After haven uploaded, don't give me any response yet and wait for my next prompt."
2) Only prompt the PUT as a single string
3) Only prompt the JUnit test as a single string
4) "Please improve the assertions in the unit test such that it will result into a higher mutation score. You don't have to explain which changes you made."

3.3 Test Sanitization

In order to use the augmented unit tests, we must first assure that all tests compile and pass when run. While the majority of tests cases generated during this study shows no errors, less than 10% of all test cases does require intervention. These errors consist of one or multiple incorrect assertion(s) per test case, and on rare occasions due to exceptions within the assertion statement (<1%). Out of these exception errors, all but two are null pointer exceptions, the remainder being a single number format exception and single non-existing method call, caused by hallucination of the LLM. To sanitize the generated test cases, we manually remove assertions showing such effects. In case this causes a test case to not have any assertions, the entire test case is removed.

4 EMPIRICAL STUDY

To assess the impact of evoLLve'M on improving JUnit test assertions and mutation score, we perform an empirical evaluation to answer the following research questions:

RQ1: *How effective are the resulting tests from evoLLve'M compared to only using Evosuite, measured by mutation score?*

RQ2: *What is the impact of evoLLve'M on types of mutation killed, compared to only using Evosuite?*

4.1 Benchmark

For evaluation of the proposed approach, we have assessed it on a benchmark consisting of a diverse set of classes, gathered from the SF110 Corpus. Besides the used package and class name, the cyclomatic complexity of each PUT is also included, as shown in Table 2. We motivate the choice of this benchmarks because it is not listed on well known developer platforms such as GitHub nor used frequently in research. Hence with high likelihood, the classes are not included in the training set for ChatGPT-4o, which could skew the results as mentioned prior.

4.2 Configurations and Parameter Settings

In this research, EvoSuite is used as version V1.2.0 with the DynaMosa algorithm on a fixed search budget of 120 seconds, set to weak mutation coverage. While additional parameter tuning has an impact on the performance of the algorithm, used default settings provide a valid results [1]. All initial JUnit tests generations were performed on two AMD EPYC 7H12 64-Core processors, resulting into 128 and 512 threads with hyper-threading enabled. Here, the CPU operates on a clock speed between 1.5GHz and 2.6GHz, with 256GB of RAM-memory using Ubuntu 22.04 as operating system.

Considering ChatGPT, we have used version gpt-4o-2024-05-13. All tests have been augmented on the 16th and 17th of June, 2024. No specific seeds have been used during this process.

Last, all operations regarding mutation testing are performed using PIT version 16.1.1. All types of applied mutations are listed in Table 1, together with a description and examples. These types form the default setting of PIT, specifically targeting weak mutation, being representative for a valid assessment [17].

4.3 Analysis

Both approaches used in this study are non-deterministic, implying different runs can lead to variying results. To mitigate this issue, both EvoSuite and evoLLve'M are run 6 times for each of the 20 classes. In order to address the variance of using EvoSuite generated tests as an input for the LLM, only the most representative test with regards to median mutation score over all runs has been used for each class. To measure the relative performance, we apply a unpaired Wilcoxon signed-rank test, with a significance level of 0.05. This means that only for p-values equal or smaller than 0.05, both approaches show a statistically significant difference. Furthermore, the Vargha-Delaney $\hat{A}_{12}$ statistic is utilized to measure the actual effect size (i.e., magnitude) of any significantly distinct finding.

5 RESULTS

This section analyzes the results of the conducted experiments to answer the research questions stated in section 4. Table 2 summarizes the results of the comparison between EvoSuite (i.e., the baseline) and evoLLve'M for research question 1. For each class, we show the median mutation score, the statistical significance produced by the Wilcoxon test, and the effect size using the Vargha-Delaney statistic. In the table, we denote the p-value for classes having an equal distribution in mean mutation score compared to

Table 1: List of default Pitest mutatio types used for mutation testing

Operator	Description
CB - conditional boundary	replaces relational operators with their boundary counterpart (i.e., < with <=)
ER - empty returns	replaces return values for primitive types with an empty value (i.e., 0 for int, "" for strings)
FR - false returns	replaces primitive and boxed boolean return values with false
TR - true returns	replaces primitive and boxed boolean return values with true
INC - increments	interchanges increment and decrement operations of local variables (i.e., i++ with i- -)
IN - invert negatives	1 in 40,000
MTH - math	replaces binary arithmetic operations for integers and float-points (i.e., + with -, » with «)
NC - negated conditionals	replaces conditionals with their negation (i.e. == with !=, < with >=)
NR - null returns	replaces method return values with null objects
PR - primitive returns	replaces int, short, long, char, float and double return values with 0
VMC - void method call	removes calls to void methods

the baseline, with "-", as well as for classes with a negligible effect size. Results showing a statistically significant improvement are highlighted in gray. In order to give a valid representation of the effectiveness of both approaches for the entire test suite, uncovered mutations or mutations leading to timeouts have not been excluded. For the second research question, Table 3 summarizes the results of the comparison for both approaches, now for all mutation types that were killed during the experiments. To emphasize the results in killed mutations rather than test suite effectiveness, uncovered mutations or mutations leading to timeouts have been now been removed.

5.1 Research Question 1

As shown in Table 2, our approach achieves significantly higher mutation scores than EvoSuite for 5 out of 20 classes, all having a large effect size. The biggest gain of 39.4% is achieved for the ByteVector class, whereas the smallest gain was 8.64% for util.Queue. Furthermore, we notice that 4 classes initially reached full mutation score before augmentation, which also remained equal afterwards. Since this does not contribute to any benefit for comparison, removing these classes would rather imply a significant improvement for 31.3% out of all classes, rather than only 25%.

In summary, our findings indicate that evoLLve'M achieves a **higher or equal mutation score** as compared to only using EvoSuite. For classes with no improvement in mutation score, evoLLve'M does not show a significant negative impact.

5.2 Research Question 2

As shown in Table 3, while evoLLve'M does achieves a 3% increase in total mutations killed compared to EvoSuite, both approaches achieve similar results over most mutation types in total. However, evoLLve'M achieves a 26.9% increase in mutations killed for the increment (INC) type, as well as a 8.9% increase for the null return (NR) type.

In summary, our findings indicate that evoLLve'M achieves a **higher or similar amount of killed mutations per type** for weak mutation, as compared to only using EvoSuite.

6 THREATS TO VALIDITY

In this section, we will address possible threats to the validity of the study and the measures taken to reduce their impact.

The first threat is reproducibility of the study. LLMs are non-deterministic systems, which implies output variability [12], even if the same set of prompts are being used. In over to overcome this, we have repeated the generation of augmented unit tests over 6 runs, where we take the median mutation score to account for variability in the results. Furthermore, we make use of a closed source LLM in this study, which gives no control over adaptions that are made to the model over time. This is called model evolution unpredictability [12], and gives no assurance that the results will remain consistent over time. To address this measure, we included the specific model using during the study, as well as on which dates it was used.

The second threat is implicit data leakage [12], implying we cannot guarantee which data was used during pre-training of the LLM, which would lead to an unfair advantage in the results. However, the SF110 Corpus is not listed on well known developer platforms such as GitHub, and only infrequently used in research. This motivates its use during the study, mitigating the effect of training that could skew the results.

The last threat comes from internal validity, reflecting on the accuracy of the study, specifically for augmenting JUnit tests. Since the scope of the research was limited in time, we utilized only the median EvoSuite generated test with regards to mutation score, as an input for test augmentation. Due to the variability in the resulting EvoSuite tests, using multiple tests as input for the same class would lead to more complete results. Nevertheless, we still use an input that represents the median over 6 different.

7 RESPONSIBLE RESEARCH

In this section, we address the societal and ethical impact of the study, the first being reproducibility, also mentioned as a threat to validity. The reproducibility aspect provides transparency and credibility for further research purposes, allowing others to correct understanding of the work that was done. To ensure the extent of reproducibility, multiple measures have been taken. First, all used tools used during the study have been listed, containing the version, and if applicable also the date of use and information about the machine it was run on.

Table 2: Median mutation score

ID	Project Name	Program Under Test	CC	EvoSuite	evoLLve'M	p-value	12
1	Tullibee	client.Util		1.00	1.00	-	- (0.5)
2	Tulibee	client.Contract		1.00	0.99	0.26	S (0.36)
3	templateit	OpMatcher		0.72	0.94	0.04	L (0.92)
4	sfms	crypt.Base64		0.78	0.78	0.85	- (0.5)
5	imSMART Migration	servlet.HTMLFilter		1.00	1.00	-	- (0.5)
6	BeanBin	search.WildcardSearch		0.96	0.96	-	- (0.5)
7	saxpath	Axis		1.00	1.00	-	- (0.5)
8	Java View Controller	tools.Base64Coder		0.95	0.96	0.26	M (0.72)
9	Java View Controller	tools.HtmlEncoder		0.73	0.67	0.08	L (0.25)
10	Corina	util.NaturalSort		0.67	0.66	0.84	- (0.44)
11	Corina	util.Sort		0.07	0.19	0.10	L (0.75)
12	Corina	util.StringComparator		1.00	1.00	-	- (0.5)
13	Corina	util.StringUtils		0.88	0.89	0.71	S (0.63)
14	SchemaSpy	util.Version		0.84	0.95	0.03	L (0.92)
15	Java Interactive Profiler	ByteVector		0.33	0.46	0.03	L (1.00)
16	Lagoon	util.Utils		0.96	0.96	0.16	M (0.67)
17	openjms	util.CommandLine		0.88	0.84	0.41	M (0.71)
18	biblestudy	util.Queue		0.81	0.88	0.04	L (0.94)
19	Battlecry	bcWord		0.78	0.87	0.03	L (0.83)
20	fim1	utils.StringEncoder64		0.77	0.71	0.06	M (0.28)
Mean over all projects				80.6%	83.5%		

Table 3: Evaluation of killed mutants over all runs

Type	EvoSuite		evoLLve'M	
	killed	total	killed	total
CB	359 (67.2%)	534	370 (69.3%)	534
ER	249 (90.2%)	276	260 (93.9%)	277
FR	90 (100%)	90	88 (100%)	88
TR	199 (97.1%)	205	202 (97.1%)	208
INC	290 (65.3%)	444	368 (82.9%)	444
MTH	1107 (76.0%)	1456	1082 (75.1%)	1440
NC	1529 (90.69%)	1686	1565 (91.4%)	1712
NR	133 (88.7%)	150	146 (97.3%)	150
PR	241 (84.9%)	284	249 (85.3%)	292
VMC	189 (79.4%)	238	188 (79.7%)	236
Total	**4386 (81.8%)**	5363	**4518 (84.0%)**	5381

Furthermore, another ethical impact of the study is on security and privacy. This could lead from the use of OpenAI's ChatGPT-4o model, as recent work has shown suspicion about the origin of data used during pre-training of LLMs [28]. While we do not encourage the unauthorized use of private data, there is little impact we can make besides using a different model.

8 CONCLUSION AND FUTURE WORK

In this paper, we have leveraged search-based testing and test generation using an LLM to improve assertions in JUnit tests, with the aim of achieving a higher mutation score. We implemented our approach using EvoSuite and ChatGPT-4o, and evaluated it on a benchmark consisting of 20 Java classes. The results show that evoLLve'M significantly improves mutation score, confirming our initial hypothesis. Furthermore, it shows research focusing on the intersection of both utilized approaches can be beneficial, rather than only focusing on one.

In future work, we plan to confirm current findings by evaluating the proposed approach on a larger set of classes, on different search budgets for EvoSuite. This can give more insight in which types of classes are most positively impacted, while potentially allowing for cheaper operating costs. Furthermore, generating the unit tests with EvoSuite already generates logs containing meta-data on the killed and survived mutations for these test. This could be of large benefit during prompt augmentation, but was not used as the tool does not provide features to efficiently extract this data yet. A further next step is to adapt the approach to work with focal methods and unit test cases, rather than providing the entire PUT and JUnit test, as this has also shown positive results in recent literature, leading to a decrease in hallucinations.

REFERENCES

[1] Andrea Arcuri and Gordon Fraser. 2011. On Parameter Tuning in Search Based Software Engineering. 33–47. https://doi.org/10.1007/978-3-642-23716-4_6

[2] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[3] Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. 2015. When, how, and why developers (do not) test in their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 179–190. https://doi.org/10.1145/2786805.2786843

[4] José Campos, Yan Ge, Nasser Albunian, Gordon Fraser, Marcelo Eler, and Andrea Arcuri. 2018. An empirical evaluation of evolutionary algorithms for unit test suite generation. *Information and Software Technology* 104 (2018), 207–235. https://doi.org/10.1016/j.infsof.2018.08.010

[5] A. Dakhel, A. Nikanjam, V. Majdinasab, F. Khomh, and M. Desmarais. 2024. Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing. https://arxiv.org/pdf/2308.16557

[6] Khalid El Haji, Carolin Brandt, and Andy Zaidman. 2024. Using GitHub Copilot for Test Generation in Python: An Empirical Study. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024) (AST '24)*. Association for Computing Machinery, New York, NY, USA, 45–55. https://doi.org/10.1145/3644032.3644443

[7] Danielle Gonzalez, Joanna C.S. Santos, Andrew Popovich, Mehdi Mirakhorli, and Mei Nagappan. 2017. A Large-Scale Study on the Usage of Testing Patterns That Address Maintainability Attributes: Patterns for Ease of Modification, Diagnoses, and Comprehension. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 391–401. https://doi.org/10.1109/MSR.2017.8

[8] S. Hossain and M. Dwyer. 2024. TOGLL: Correct and Strong Test Oracle Generation with LLMs. https://arxiv.org/html/2405.03786v1

[9] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. New York, NY, USA, 435–445. https://doi.org/10.1145/2568225.2568271

[10] Gunel Jahangirova and Valerio Terragni. 2023. SBFT Tool Competition 2023 - Java Test Case Generation Track. In *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. 61–64. https://doi.org/10.1109/SBFT59156.2023.00025

[11] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. https://doi.org/10.1109/TSE.2010.62

[12] Thomas Durieux June Sallou and Annibale Panichella. 2024. *Breaking the Silence: the Threats of Using LLMs in Software Engineering.* NewIdeasandEmergingResults(ICSE-NIER,2024)

[13] Divya Kumar and K.K. Mishra. 2016. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Computer Science* 79 (2016), 8–15. https://doi.org/10.1016/j.procs.2016.03.003 Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[14] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K. Lahiri, and Siddhartha Sen. 2023. CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 919–931.

[15] Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024. Calibrating LLM-Based Evaluator. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (Eds.). ELRA and ICCL, Torino, Italia, 2638–2656. https://aclanthology.org/2024.lrec-main.237

[16] Phil McMinn. 2004. Search-based software test data generation: a survey: Research Articles. *Softw. Test., Verif. Reliab.* 14 (06 2004), 105–156. https://doi.org/10.1002/stvr.294

[17] A. Jefferson Offutt and Stephen D. Lee. 1991. How strong is weak mutation?. In *Proceedings of the Symposium on Testing, Analysis, and Verification* (Victoria, British Columbia, Canada) *(TAV4)*. Association for Computing Machinery, New York, NY, USA, 200–213. https://doi.org/10.1145/120807.120826

[18] Mitchell Olsthoorn, Arie van Deursen, and Annibale Panichella. 2020. Generating Highly-structured Input Data by Combining Search-based Testing and Grammar-based Fuzzing. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1224–1228.

[19] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435

[20] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. A large scale empirical comparison of state-of-the-art search-based test case generators. *Information and Software Technology* 104 (2018), 236–256. https://doi.org/10.1016/j.infsof.2018.08.009

[21] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Chapter Six - Mutation Testing Advances: An Analysis and Survey. Advances in Computers, Vol. 112. Elsevier, 275–378. https://doi.org/10.1016/bs.adcom.2018.03.015

[22] James Shore and Shane Warden. 2007. *The art of agile development* (first ed.). O'Reilly.

[23] Mohammed Latif Siddiq, Joanna Cecilia Da Silva Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinícius Carvalho Lopes. 2024. Using Large Language Models to Generate JUnit Tests: An Empirical Study. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 313–322. https://doi.org/10.1145/3661167.3661216

[24] Claudio Spiess, David Gros, Kunal Suresh Pai, Michael Pradel, Md Rafiqul Islam Rabin, Amin Alipour, Susmit Jha, Prem Devanbu, and Toufique Ahmed. 2024. Calibration and Correctness of Language Models for Code. arXiv:2402.02047

[25] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. 2023. ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation. arXiv:2307.00588

[26] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. 2024. ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation. *IEEE Transactions on Software Engineering* 50, 6 (2024), 1340–1359. https://doi.org/10.1109/TSE.2024.3382365

[27] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, Qing, and Wang. 2024. *Software Testing with Large Language Models: Survey, Landscape, and Vision.* https://arxiv.org/pdf/2307.07221

[28] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly. *High-Confidence Computing* 4, 2 (2024), 100211. https://doi.org/10.1016/j.hcc.2024.100211

[29] Peng Zhang, Yang Wang, Xutong Liu, Zeyu Lu, Yibiao Yang, Yanhui Li, Lin Chen, Ziyuan Wang, Chang-Ai Sun, Xiao Yu, and Yuming Zhou. 2024. Assessing Effectiveness of Test Suites: What Do We Know and What Should We Do? *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 86 (apr 2024), 32 pages. https://doi.org/10.1145/3635713

[30] Yucheng Zhang and Ali Mesbah. 2015. Assertions are strongly correlated with test suite effectiveness. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 214–224. https://doi.org/10.1145/2786805.2786858