Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 46 (2025), No. 4-B, pp. 1347–1356
DOI : 10.47974/JIOS-2001

AI-driven green testing : Optimizing efficiency and sustainability in software testing

Manoj Kumar Singhal *
Lead Software Engineer
Opaque Systems
California
United States

Chhaya Gunawat †
System Development Engineer
Amazon
California
United States

Abstract

Software has become integral to daily life, continually expanding with increasingly complex features to meet growing expectations. However, managing these complexities—from understanding application dependencies to ensuring reliability through rigorous testing—poses significant challenges.

For organizations, delivering robust and dependable software is crucial for maintaining business success and reputation. Therefore, a significant portion of the software development life cycle is dedicated to testing. Engineers often write thousands of test cases to validate new features and prevent regressions. Despite these efforts, current regression testing methods are inefficient, as they often require developers to run all test cases irrespective of whether the code paths have been altered.

This paper proposes an AI-driven approach to optimize regression testing. By analyzing specific trends, the AI identifies and prioritizes the most relevant test cases, thereby reducing execution time and resource consumption. This approach promises to mitigate inefficiencies associated with traditional testing methods, offering a more cost-effective and timely solution for software development cycles.

Subject Classification: *68M15.*

Keywords: *AI-driven testing, Regression test optimization, Test case prioritization, Cost effective testing, Green testing.*

* *E-mail:* **manojsinghal258@gmail.com (Corresponding Author)**
† *E-mail:* **chhayagunawat@gmail.com**

1. Introduction

Software development has become increasingly sophisticated, with applications incorporating complex features to satisfy the growing demands of users. Ensuring the reliability and robustness of such software necessitates rigorous testing, a critical component of the software development lifecycle (SDLC). Traditional regression testing methods, while thorough, often lead to inefficiencies by requiring the execution of all test cases, regardless of recent code changes.

In a typical large-scale enterprise with a distributed, multi-tenant environment, the number of regression test cases run depends heavily on the software application and deployment environment. Nightly runs typically involve 5,000 to 10,000 test cases, taking 12 to 24 hours, while weekly runs range from 40,000 to 60,000 test cases and require a couple of days to complete, ensuring release stability.

In such deployment environments, this testing practice consumes approximately 40-50% of resources for basic sanity testing. Despite the significant resource allocation, these tests seldom change and yield minimal results.

The Need for Efficient Regression Testing Regression testing is crucial for ensuring that new code changes do not compromise existing functionality. Engineers often develop comprehensive test suites to encompass various scenarios and edge cases. However, running all these tests can become redundant when significant portions of the code remain unchanged. This inefficiency not only delays the development cycle but also increases testing costs.

Current Challenges in Regression Testing

- **Time Consumption:** Running a full suite of regression tests is time-consuming, often leading to bottlenecks in the development pipeline.
- **Resource Intensive:** Full regression testing requires substantial computational resources, impacting overall productivity and increasing operational costs.
- **Redundancy:** Many test cases may not be relevant to the recent changes, leading to unnecessary test executions.

2. Related Work

Software engineering encompasses more than just programming and development. It involves the systematic implementation of engineering procedures throughout the software development lifecycle. Within this process, software testing stands out as a phase that requires substantial time for execution and often represents a significant portion of the development costs [1].

Interoperability/Regression testing ensures new changes don't disrupt existing software, challenges include third-party components and agile development process. Instead of retesting all the cases, techniques like test suite minimization, test case selection, and prioritization can be used for efficiency. These methods are cost-effective, further research in this area can improve these techniques [2].

Darko Marinov and his team [3] utilized STARTS, a Maven plugin designed for Java projects, to perform Static Regression Test Selection. This tool operates without requiring code instrumentation and constructs a dependency graph. Their findings indicated that STARTS selects 35.2% of tests, leading to an 19% reduction in end-to-end regression time[4].

Test case prioritization orders test cases to improve performance goals like fault detection rate, which measures how quickly faults are found. Faster detection gives quicker feedback. Test cases are prioritized by criteria such as early fault detection and execution time. This technique enhances testing efficiency by providing faster feedback on the system under test[5].

Gartner highlights that nearly half of software engineering leaders prioritize customer or user satisfaction. This delves into the pivotal role of regression testing, addressing its challenges and exploring effective strategies to overcome them. It emphasizes the significant impact of automation and examines the evolving landscape where AI promises to revolutionize regression testing. Regression Testing being one of the major testing types requires much time when done manually. It typically tests whether the software or the application works properly after the fixation of any bugs or errors [6]. Each strategy offers unique advantages and drawbacks, emphasizing the need for tailored solutions based on factors like code complexity, available resources, and quality expectations. The integration of AI into regression testing processes holds potential to accelerate testing and enhance overall software quality as part of a continuous quality strategy[7].

Based on the above studies, we have observed that in recent years, the field of regression testing has seen significant advancements, particularly with the integration of AI and machine learning techniques to optimize test case selection and prioritization. Several studies and surveys provide a comprehensive overview of these developments.

These studies collectively underscore the potential of AI and machine learning in transforming regression testing practices. By incorporating advanced algorithms and analytical techniques, these approaches aim to streamline the testing process, reduce redundancy, and enhance the overall efficiency of software development cycles.

3. Proposed Work

In recent years, artificial intelligence (AI) has proven to be a powerful technology for analyzing complex datasets and achieving desired results, as evidenced by its successful applications in industries such as autonomous driving. To tackle the challenge of efficiently selecting regression test cases on a daily basis, given their large scale and complexity, we propose leveraging AI to make informed decisions. The EC2 Cloud makes it easier for users to set up and run Hadoop applications on a large-scale virtual cluster [8].

AI-Driven Approach: The AI-driven approach involves analyzing code changes, historical test data, and execution trends to prioritize the most relevant test cases. This method focuses on:

1. **Risk/Change Analysis:** Using machine learning algorithms to identify patterns and trends in code changes and their impact on different parts of the software. During component-based software development can be overcome by sharing relevant component information required for RTS analysis [9].

For example, as shown in table 1, tracking all the file changes, their dependent files and relevant test cases.

Table 1

Tracking the Code Changes

File Name	Code Line Changed	Test Case ID	dependent
foo.c	#105-115,	TC101-T C119	bar.c
foo.c	#176-189	TC201-T C220	

Contd...

bar.c	Nil	TC801-T C830	ab.c
ab.c	Nil	TC901-T C902	

2. **Test Case Prioritization:** Based on the above analysis, the model can select a subset of test cases for regression testing. Digital transformation has moved behaviors from traditional business to IT-driven business [10]. To further enhance execution efficiency, it dynamically prioritizes test cases that are most likely to detect regressions, leveraging historical trends. For instance, it may prioritize test cases that historically failed when similar changes were made, as explained in table 2.

Table 2
Test Case Prioritization

Test Case ID	Historical Execution Results				
	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run N**
TC1	Pass	Pass	Pass	Pass	Pass
TC2	Fail	Fail	Fail	Fail	Fail
TC3	Pass	Fail	Pass	Fail	Pass
TC4					
-					
TC N					

3. **Resource Optimization:** This approach prioritizes executing the most common test cases first, then it picks the sanity tests and then as per the coverage.

By doing so, it aims to reduce the total number of tests run, conserve computational resources, and minimize execution time.

3.1 *Implementation Methodology*

As explained in fig.1, the implementation of the AI-driven regression testing optimization involves several key steps:

1. **Data Collection:** Gathering historical test execution data, code change logs, and failure reports.

2. **Feature Extraction:** Identifying features from the collected data that influence test outcomes, such as code complexity, change frequency, and past test results.
3. **Model Training:** Training machine learning models on the extracted features to predict the relevance of test cases for new code changes.
4. **Test Case Selection:** Applying the trained models to prioritize and select test cases for execution.

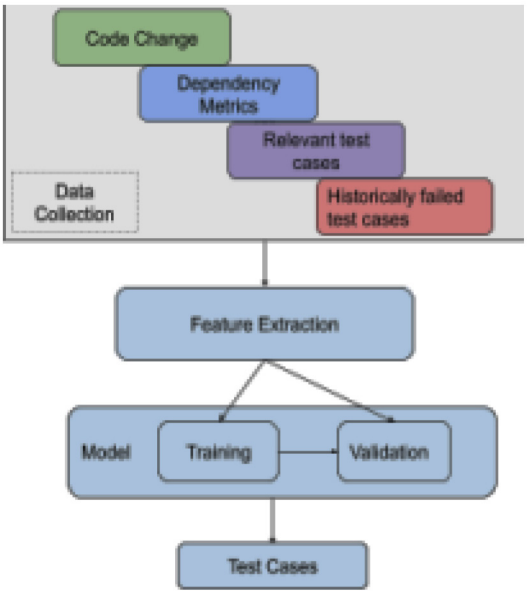

Figure 1
Execution Workflow

3.2 *Model Selection*

Choosing the appropriate model is essential for effective training and achieving optimal results. Following are a few potential models.

Classification Models

- **Logistic Regression:** For binary relevance prediction.
- **Random Forest:** For handling complex interactions between features.
- **Gradient Boosting:** For improved accuracy through boosting techniques.

Ranking Models

- **Learning to Rank (LTR):** For prioritizing test cases based on relevance.
- **Bayesian Network Ranking (BNR)**: Uses probabilistic graphical models to represent and solve ranking problems.

3.3 *Model Training*

The next step is to train the model on the relevant data and validate it. The following is the training process.

1. **Data Splitting:** Split the data into training, validation, and test sets.
2. **Model Training:** Train the selected model(s) using the training set and use cross-validation to tune hyperparameters and prevent overfitting. 3. **Evaluation Metrics:**

- Precision, Recall, and F1-Score: To evaluate the relevance of selected test cases.
- Mean Reciprocal Rank (MRR): For ranking performance.
- Test Coverage: Ensuring that the critical paths are tested.

3.4 *Model Evaluation and Validation*

Once the model is trained, the next step is to evaluate it for accuracy before integrating it into a continuous deployment environment. The following are a few evaluation techniques:

1. **Holdout Validation:** Evaluate the model on a holdout test set to ensure generalization.
2. **Cross-Validation:** Use k-fold cross validation to assess model performance and stability.
3. **Performance Metrics:** Calculate metrics like execution time reduction, resource savings, and defect detection rate.

3.5 *Deployment and Monitoring*

Once the model has been evaluated then it needs to be integrated into the deployment. The following are the steps:

1. **Model Integration**:
   a. Integrate the trained model into the CI/CD pipeline.

   b. Implement a mechanism to automatically select and prioritize test cases based on model predictions.

2. **Monitoring**
   a. Continuously monitor the performance of selected models.
   b. Collect feedback on the relevance of selected test cases and use it for model retraining.

3. **Retraining**
   a. Periodically retrain the model with new data to maintain accuracy and relevance.
   b. In case of re-designing the feature, the model should be un-train and train.

This outline covers the essential steps to design and implement an AI-driven model for optimizing regression testing. Adjustments may be necessary based on specific project requirements and data availability. Machine learning (ML), artificial intelligence (AI), and adaptive platforms continuously change underlying software and thus prohibit traditional coverage schemes and functional testing [11].

4. Risk Associated

Implementing AI-driven approaches to optimize regression testing involves navigating several critical risks. These include ensuring the quality and availability of data, guarding against model overfitting, enhancing the interpretability of models, addressing scalability and performance concerns, managing integration challenges effectively, and mitigating biases in model predictions. Each of these risks poses potential obstacles that must be carefully assessed and managed to ensure the success and reliability of AI-driven regression testing implementations. By proactively addressing these challenges, organizations can harness the full potential of AI while minimizing potential drawbacks and ensuring robust testing outcomes.

5. Conclusion

In this paper, we proposed an AI-driven approach to optimize regression testing by prioritizing relevant test cases, thus addressing the inefficiencies associated with traditional regression testing methods [12]. By leveraging machine learning algorithms, our approach analyzes code

changes, historical test data, and execution trends to select and prioritize test cases most likely to detect regressions. This method not only reduces execution time and resource consumption but also enhances the agility and cost-effectiveness of the software development lifecycle. Our results demonstrate the potential of AI-driven regression testing to significantly improve testing efficiency, making it a viable solution for organizations seeking to streamline their development processes.

6. Future work

Future work in this domain could explore several avenues to further enhance the AI-driven regression testing approach. One area of interest is the integration of more sophisticated machine learning models and deep learning techniques to improve the accuracy and relevance of test case selection. Additionally, developing a more comprehensive dataset that includes various types of software applications and testing scenarios could further validate and generalize the proposed method. Another potential direction is the incorporation of real-time feedback mechanisms to continuously refine and adapt the AI models based on ongoing test results. Finally, expanding the approach to include automated test case generation and repair could further streamline the regression testing process, making it more robust and adaptive to evolving software development needs.

References

[1] M. Khatibsyarbini, M. H. H. A. Bakar, M. Z. I. Abdul Karim, and M. I. Ghani, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology,* vol. 93, pp. 74–93 (2018).

[2] F. Martinelli and I. Matteucci, "A framework for automatic generation of security controller," *Software Testing, Verification and Reliability,* vol. 22, no. 8, pp. 563–582 (Dec. 2012).

[3] O. Legunsen, A. Shi, and D. Marinov, "STARTS: STAtic regression test selection," in Proc. 32nd IEEE/ACM Int. Conf. Automated Software Engineering (ASE), Urbana-Champaign, IL, USA, pp. 949–954 (2017).

[4] O. Legunsen, A. Shi, and D. Marinov, "STARTS: STAtic regression test selection," in Proc. 32nd IEEE/ACM Int. Conf. Automated Soft-

ware Engineering (ASE), Urbana-Champaign, IL, USA, pp. 949–954 (2017).

[5] K. Kaur and V. Chopra, "Test case prioritization using genetic algorithm," *International Journal of Computer Science and Mobile Computing,* vol. 5, no. 6, pp. 488–494 (2016).

[6] A. K. Arumugam, "Software testing techniques & new trends," *International Journal of Engineering Research & Technology* (IJERT), vol. 8, no. 12 (2019).

[7] R. Khankhoje, "AI in test automation: Overcoming challenges, embracing imperatives," *International Journal on Soft Computing, Artificial Intelligence and Applications* (IJSCAI), vol. 13, no. 1, pp. 1–10 (2024).

[8] M. Khan, A. Ghafoor, J. W. Byun, and M. S. Kim, "Hadoop performance modeling for job estimation and resource provisioning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 441–454 (Feb. 2015).

[9] S. Biswas, R. Mall, A. Mohanty, and S. Sukumaran, "Regression test selection techniques: A survey," *Informatica*, vol. 35, no. 3, pp. 289–321 (2011).

[10] R. Verdecchia, A. Procaccianti, and P. Lago, "Green IT and green software," *IEEE Software,* vol. 38, no. 6, pp. 7–15 (Nov./Dec. 2021).

[11] C. Ebert, D. Bajaj, and M. Weyrich, "Testing software systems," *IEEE Software*, vol. 39, no. 4, pp. 8–17 (Jul./Aug. 2022).

[12] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120 (Mar. 2012).

Received October, 2024