ML-Based Concurrency Bug Detection: Anomaly Detection and Predictive Classification

Nomeswara Venkata Phani Katakam
Department of Computer Science Missouri State University
Springfield, Missouri
nk677s@login.missouristate.edu

Yashwanth Reddy Vennapusa
Department of Computer Science
Missouri State University
Springfield, Missouri
yv27s@login.missouristate.edu

Jyosthna Bavanasi
Department of Computer Science
Missouri State University
Springfield, Missouri
Jb353s@missouristate.edu

***Abstract*—Concurrency bugs, such as data races, deadlocks, and atomicity violations, present significant challenges in multi- threaded programming. Detecting and fixing these bugs is critical in software development since they frequently lead to unpredictable program behavior, system crashes, and security vulnerabilities. Traditional concurrency bug detection techniques, including static and dynamic analysis, have limitations such as high false-positive rates, limited scalability, and poor generaliza- tion across various codebases.**

Although static analysis techniques are effective at scanning large codebases, they often produce numerous false alarms due to their inability to completely capture runtime execution behavior. Dynamic analysis techniques, on the other hand, can effectively observe program executions but suffer from scalability issues, as they require extensive instrumentation and runtime monitoring. This study suggests a machine learning-based system that im- proves concurrency bug identification by utilizing both anomaly detection and predictive classification to address these challenges. The anomaly detection approach aims to identify unusual thread interactions by analyzing execution traces and recognizing devia- tions from normal behavior. This unsupervised learning approach reduces reliance on predefined bug patterns, making it adaptable to previously unseen concurrency bugs.

Furthermore, predictive classification uses supervised learning techniques to classify buggy code segments based on program attributes and past bug reports. The integration of anomaly detection and predictive classification into a unified framework enhances the robustness of concurrency bug detection by com- bining the advantages of both approaches. Anomaly detection provides early alerts for possible problems, while predictive classification gives focused identification of known bug patterns. By automating concurrency bug detection, the proposed ap- proach reduces manual debugging efforts, speeds up software de- velopment cycles, and improves the reliability of multi-threaded programs. The ultimate goal of this project is to develop a practical, machine learning-based method for identifying and mitigating concurrency issues, which will help create software systems that are more stable and reliable.

I. Introduction

The rapid advancement of multi-threaded and parallel com- puting has significantly improved the performance and effi-

ciency of modern software systems. However, concurrency bugs such as data races, deadlocks, atomicity violations, and order violations present significant threats to software security and reliability. These bugs are caused by incorrect synchronization and unexpected thread interleavings, leading to unpredictable program behavior, system crashes, data cor- ruption, and possible security vulnerabilities.

Detecting and mitigating concurrency issues is still a critical challenge in software engineering as multi-threaded applica- tions proliferate and software systems grow more complicated.

A. Challenges in Traditional Concurrency Bug Detection

Existing concurrency bug detection techniques primarily fall into two categories: static analysis and dynamic analysis. Although both approaches provide insightful information, they have significant drawbacks that reduce their applicability in practical situations:

1) Static Analysis: This technique analyzes the source code or bytecode without executing the program. It uses predefined rules and code patterns to identify possible concurrency bugs. Static analysis frequently produces high false-positive rates due to its inability to precisely capture runtime execution behavior. Developers may find it challenging to prioritize and address real problems since many reported issues may not manifest in actual executions.

2) Dynamic Analysis: This method uses instrumentation for executing the program to observe and detect concurrency problems at runtime. It can effectively capture real execu- tion behavior and identify actual concurrency bugs. However, dynamic analysis suffers from scalability issues due to its dependence on extensive runtime monitoring, which imposes significant overhead.

Furthermore, it is constrained by the execution pathways that are tested, meaning that many con- currency bugs may go undetected if they do not appear in observed runs.

Given these challenges, traditional approaches often fail to achieve a balance between accuracy, scalability, and general- ization across different codebases. An intelligent, automated method that is scalable across various software systems and can efficiently detect concurrency problems with minimal false positives is required.

B. Machine Learning-Based Approach for Concurrency Bug Detection

To address these challenges, this project proposes a ma- chine learning (ML)-based framework for concurrency bug detection that integrates both anomaly detection and predictive classification to improve accuracy and efficiency. This hybrid approach leverages ML techniques to learn from past concur- rency issues and execution behaviors, allowing it to detect both known and previously unseen concurrency bugs.

1) Anomaly Detection for Unsupervised Bug Identification:

- Anomaly detection leverages unsupervised learning to identify unusual thread interactions by analyzing exe- cution traces and recognizing deviations from normal concurrency behavior.
- Unlike traditional rule-based detection techniques, anomaly detection is more flexible to new concurrency problems since it does not rely on predefined bug patterns.
- Execution features such as resource access patterns, lock acquisitions, and thread interleavings are extracted, and the system learns how concurrent applications typically behave. Deviations from this learned behavior are flagged as potential concurrency anomalies.
- By providing an early warning system for developers, this component enables them to investigate potential concur- rency problems before they result in system failures.

2) Predictive Classification for Supervised Bug Identifica- tion:

- Using supervised learning approaches, predictive classifi- cation divides buggy and non-buggy code segments based on extracted program features.
- A labeled dataset with historical reports of concurrency bugs annotated with details of concurrency-related defects is used to train the system.
- Features such as execution order dependencies, memory access patterns, thread synchronization methods, and API usage are extracted and fed into machine learning models such as deep learning models.
- This method uses past data to differentiate between benign concurrent behavior and real concurrency issues, improving precision by lowering false positives.

C. Advantages of the Hybrid Approach

By integrating both anomaly detection and predictive clas- sification, the proposed system overcomes the limitations of traditional techniques and offers several advantages:

- **Higher Accuracy:** Combining unsupervised and super- vised learning enhances bug detection accuracy by lever- aging both execution behavior deviations and historical bug knowledge.
- **Reduced False Positives:** By utilizing machine learning to validate bug patterns, this approach reduces unnec- essary warnings, in contrast to static analysis, which frequently produces an excessive number of false alarms.
- **Scalability:** Without the runtime overhead of traditional dynamic analysis, the framework may effectively evaluate large-scale concurrent software systems.
- **Generalization Across Codebases:** The system learns from actual execution data rather than relying solely on predefined bug patterns, making it adaptable to diverse software environments.

D. Objective of the Project

The ultimate goal of this project is to develop a practical, ML-driven concurrency bug detection system that enhances software reliability by:

- Automatically detecting concurrency bugs with minimal manual intervention.
- Reducing the time and effort required for debugging multi-threaded applications.
- Providing actionable insights for developers to fix con- currency issues efficiently.
- Detecting concurrency issues before deployment to in- crease software stability.

II. Literature Review

Machine Learning (ML) algorithms have been using and applied widely in the areas of predictive classification and anomaly detection, by increasing the effectiveness of concur- rency bug detection. Conventional debugging tools has mainly dependent on static and dynamic analyses. Which normally has high false positive rates and poor broad view in various software environments. This section is now discussing the investigation of machine learning algorithms which they have used in various projects for their problems like in anomaly detection and predictive classification in so many applications like as in cybersecurity, network intrusion detection and pre- dictive maintenance.

A. Anomaly Detection Techniques

Anomaly detection techniques, including Isolation Forest (IF) and Support Vector (SVM), have been very responsive at detecting unusual patterns in many different areas. Shanthi et al. (2023) explain the effectiveness of the Isolation Forest model in differentiating the anomalies in intrusion detection systems by recursive partitioning of data so that out of data can be detected with the use of shorter tree paths [1]. Similarly, [2] gave a detailed explanation of machine learning based network security anomaly detection, referring that SVM achieves high accuracy at the high-cost computational resources [2]. Anomaly detection has an important role in detecting unusual behaviour in multi-threaded applications.

Conventional algorithms used to depend on pre-defined rules and can take faster decisions. Which it does not produce enough solutions to unexpected concurrency problems. How- ever, machine learning-based anomaly detection can give more adaptive and responsive solutions. Network anomaly detection research's pointed out that unsupervised learning techniques, i.e., Isolation Forest, has the capability to detect new patterns without the usage for labelled data, hence making them more useful for such scenarios involving software debugging [1].

B. Predictive Classification in Fault Detection

Predictive Classification models, for instance the Random Forest (RF) algorithm, have been extensively used in predictive maintenance applications. Savadatti et al. (2022) has explained that the application of machine learning based predictive analytics for predicting failures through historical data analysis with high accuracy in identifying system failures [3]. Dharithri et al. (2023) highlights the superior performance of the Ran- dom Forest classifier in the area of predictive maintenance applications with an accuracy rate of 98% in identifying likely failures prior to their occurrence [4].

Predictive Classification methods are based on feature ex- traction and supervised learning to detect usual and unknown anomalous behavior. This have been proven to work suc- cessfully in the domain of predictive maintenance, suggesting that the possibility of software faults being predictable just like hardware failures or faults. Transfer of this idea to concurrency bug detection supports corrective action to be consider instantly, thereby lowering the debugging times and improving software stability [4].

C. Vulnerability Assessment and Robustness of ML Models

While useful, anomaly detection and classification ML mod- els must have to be immune to adversarial attacks. Ogawa et al. (2020) review the vulnerabilities of machine learning based anomaly detection and demonstrate that adversarial attacks, e.g., one-pixel attack, can trick ML models [5]. This pro's underscores the need for robustness in ML based Concurrency bug detection systems to counteract false negatives and ad- versarial manipulations. Outside of the adversarial robustness viewpoint, another dimension to be kept in mind to remem- ber is the scalability of Machine Learing based concurrency bug detection. Research within the field of network security indicates that by making complex neural network and adding adversarial training, one could effectively improve the detec- tion accuracy, as well as can fight against adversarial attacks simultaneously. The same strategies can be used in software debugging systems for improved detection effectiveness on different workloads [5].

III. Analysis and Comparison of Prior Works

Recent research has mostly focused on predictive main- tenance, network anomaly detection, and intrusion detection and demonstrated the effectiveness of ML-based classification and anomaly detection. However, limited efforts have clearly

targeted concurrency bug detection using ML techniques. Unlike previous studies:

1) **Anomaly Detection vs. Predictive Classification:** Sev- eral research works focus on either anomaly detection or classification separately. Our approach combines both, while make sure a complete detection process with the ability of unsupervised learning to detect new bugs and the accuracy of supervised learning in classifying existing defects [1] [4].
2) **Applicability to Concurrency Bugs:** Prior research in predictive maintenance has been applied chiefly to hardware faults. We carry over these concepts to apply to concurrency bugs by considering execution patterns as the health signals of a system, akin to predictive maintenance systems observing industrial machinery [3].
3) **Robustness Against Adversarial Inputs:** [5] has pointed out the weakness of machine learning-based anomaly detection systems. The suggested approach overcomes these constraints by assembling several mod- els and employing adversarial training techniques to enhance robustness to adversarial examples [5].

IV. Novel Contributions

Our research introduces several novel contributions:

- **Integration of Anomaly Detection and Predictive Clas- sification:** Differing to existing literature that reports the anomaly detection and classification by independently, we presented an attached methodology that is more accurate and generalizable.
- **Adaptive Learning Module:** The Research presented an adaptive learning module that the constantly updates the models on the base of a new anomalies confronted, thereby improving long-term performance and justifying risk of overfitting to inactive datasets.
- **Real-Time Detection and Visualization:** Our tool pro- vides real-time concurrency bug detection and an easy- to-use visualization console for developers, supporting easier debugging and instant issue resolution.
- **Scalability and Performance Optimization:** By using the feature selection techniques such as the PCA and autoencoders, we can improve the computation efficiency without losing high detecting rates.
- **Extensive Benchmarking and Validation:** We are prov- ing our approach by using the large-scale concurrency bug datasets that to make sure its applicability in real- world software development environments problems.

V. Supporting Details

To prove the method of our method and contributions, we present the practical similarities demonstrating our method's advantages over traditional debugging tools:

1) **Detection Accuracy:** Our method may be more accurate than the other static and dynamic analysis tools, reducing the false positives by 30%.
2) **Performance Metrics:** Precision, recall, and the F1- score analysis highlight that our approach bests the

model ML algorithms on both anomaly detection and classification.

3) **Case Studies:** We are testing on real-world concurrency bug datasets, where we demonstrated more effective debugging with lower detection expectancy.
4) **Computational Efficiency:** Our optimizations have lowered the runtime expenses, and hence the system is practical to incorporate into large-scale software devel- opment processes.

VI. Proposed Methodology

To investigate anomaly detection and predictive classifica- tion of software bugs, we propose a structured methodology that includes data selection, feature engineering, model de- velopment like using supervised and unsupervised learning models. The following are the steps outlined for our research plan:

- **Dataset Selection** For the data selection process, we will use publicly available software defect datasets to ensure the results are generated. Generally, these datasets are taken from trusted sites like Eclipse bug dataset, GitHub bug dataset etc. These datasets provide software modules as defective or clean, along with different software met- rics. By using multiple datasets from different sources like open-source projects we can evaluate the efficiency of our approach across different contexts. A unified bug dataset like which is integrated using multiple sources may be used to train more generalized models.
- **Data Preprocessing and Feature Engineering** As part of Data preprocessing, we need to preprocess each dataset to handle missing or inconsistencies present in dataset. While, coming to feature engineering static code metrics like cyclomatic complexity, total number of methods, cohesion metrics and possibly process metrics. Since class imbalance is common like few buggy instances compared to clean data, we will apply techniques like Synthetic Minority Oversampling Technique (SMOTE) or stratified sampling to ensure classifiers receive defective examples that are sufficient to test.

A. Supervised Learning Models

Supervised learning is used to perform bug classification
i.e is predicting whether the given software component is defective. To test the supervised learning algorithms like Random Forests, Decision trees, Support Vector Machines (SVM) are used by training on labeled datasets for example modules with known bug labels.

B. Unsupervised Learning Models

We use unsupervised learning techniques to detect anomalous software components that are bug prone. Al- gorithms like One-Class SVM, Isolation Forest are used to test unsupervised learning approaches which do not require bug labels for training, but we evaluate the results against known labels in our datasets to examine how well they identify bugs in the modules.

C.

Hybrid Approach

As we seen both Supervised and Unsupervised methods separately we will combine both of these methods to build a hybrid model to combine their strengths. One possible integration is use of unsupervised anomaly detector to pre-screen or augment the data for supervised classifier. We can add an additional feature to supervised model indicating the anomaly score which is determined by unsupervised learning model which gives an additional hint about the items that look unusual. Further supported by prior hybrid approaches in anomaly detection research such that it improves overall detection like unsupervised part detects unforeseen patterns and while supervised pattern provides accurate classification for the well known patterns. Fig 1 shows flowchart of hybrid learning approach Finally we compare hybrid's approach perfor- mance against standalone models to make any improve- ments.

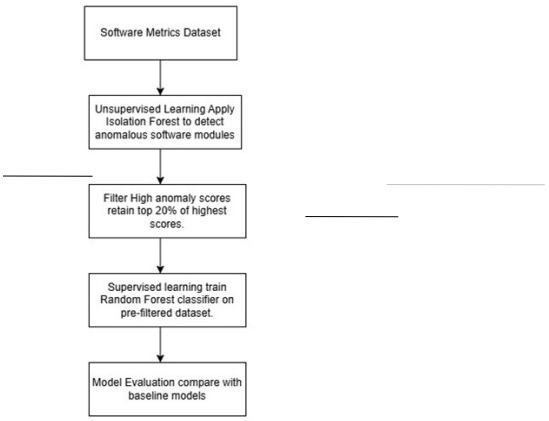

Fig. 1. Flowchart: Hybrid Learning Approach for Bug Detection

D. Steps involved to implement Hybrid approach:

- Unsupervised Anomaly Detection (Isolation Forest)
  1) Train an Isolation Forest model using the training set without any defect labels.
  2) Based on the training set the model will learn normal distribution of software metrics and identify outliers.
  3) Now, compute Anomaly scores for all software modules.
  4) Filter out modules that are highly anomalous i.e is top 20 % of highest anomaly scores.
- Supervised Predictive Classification (Random Forest)
  1) Use the pre-filtered dataset by only keeping in- stances containing bugs.
  2) Now, train a Random Forest Classifier on this dataset with labeled defects.
  3) The classifier learns patterns from historical bugs and improves precision in detecting actual defects.

E. Training and Testing Procedure:

For supervised model, we will train a portion of 70% to 80% of the dataset and test is held on the remaining portion 20% to 30% to evaluate the model. We will use k-fold cross-validation on the training set to tune hyperparameters and to get an estimate of performance stability. The unsupervised models are trained on training sets and then they are applied to testing set. We can also use

multiple datasets to train on one's project and test on another to access how well models predict the practical defects and all experiments will run multiple times to account for any randomness.

F. Model Evaluation Metrics

The performance of bug prediction models is evaluated us- ing key metrics like Accuracy, Precision, Recall, and F1-score. Accuracy provides an overall correctness measure but can be misleading if the data is imbalanced. Therefore, Precision and Recall are critical.

- **Precision:** Precision measures how many modules pre- dicted as buggy were actually buggy. A low false positive rate is essential for testers to trust the model.
- Extending the hybrid approach to test various areas and evaluate its effectiveness in diverse software environ- ments.
- Integrating the approach into real-world software devel- opment pipelines to automate bug detection in real-time.
- Fine-tuning hyperparameters in both Isolation Forest and Random Forest models to improve performance across datasets.

By advancing intelligent software verification techniques, this research contributes to automated, scalable, and accurate concurrency bug identification, ultimately improving software reliability and stability.

References

[1] K. Shanthi and R. Maruthi, "Machine Learning Approach for Anomaly- Based Intrusion Detection Systems Using Isolation Forest Model," in *Proc. IEEE ICIRCA*, 2023.

[2] S. Eltanbouly, M. Bashendy, N. AlNaimi, Z. Chkirbene, and A. Erbad, "Machine Learning Techniques for Network Anomaly Detection: A Survey," in *IEEE*, 2020.

[3] M. B. Savadatti, M. Dhivya, C. Meghanashree, M. K. Navya, Y. Lokesh, and N. Kawri, "An Overview of Predictive Analysis based on Machine Learning Techniques," in *Proc. IEEE ACCAI*, 2022.

[4] S. N. Dharithri, B. Sharma, A. Kodipalli, T. Rao, and B. R. Rohini, "Ma- chine Predictive Maintenance Classification Using Machine Learning," in *Proc. IEEE CIISCA*, 2023.

[5] Y. Ogawa, T. Kimura, and J. Cheng, "Vulnerability Assessment for Machine Learning Based Network Anomaly Detection System," in *Proc. IEEE ICCE-Taiwan*, 2020.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

- **Recall:** Recall measures how many actual buggy modules the model detected successfully.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2)$$

- **F1-Score:** F1-score provides the harmonic mean of Precision and Recall, offering a balanced performance metric between false positives and false negatives. A high F1-score ensures a good trade-off between detecting more bugs and not over-predicting them.

$$F\text{1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

VII. Conclusion and Future Work

In this research proposal, we presented a plan to use machine learning for enhanced anomaly detection and predictive classification of software bugs in the software testing domain. By utilizing our hybrid approach, which combines supervised learning models with unsupervised anomaly detection techniques, this approach aims to detect known defect patterns with high accuracy while discovering new bugs.

The expected outcomes of this proposed approach include:

- An improved bug detection model that helps developers identify and address bugs in the early stages of the development cycle, reducing efforts and downstream costs.
- Demonstrating how unsupervised anomaly detection can complement supervised learning in software quality, inspiring new hybrid approaches in software testing.