



A Comprehensive Literature Review on Automating a Bug Detector Using Machine Learning

Shagun* and Sarika

School of Computer Science Engineering and Technology,
Government College Dharamshala (H.P.), India.

(Corresponding author: Shagun*)

(Received: 27 January 2025, Accepted: 26 February 2025)

(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: Issues like bugs in software engineering are a primary cause of systems failure, regarding information security system, it is a threat. Traditional approach of bug detection system is ineffective both in its efficiency and speed. The automated bug detection system through machine learning (ML) is the scope of this study using supervised learning approaches. The study covers main parts like Data collection, preprocessing, model training, and model evaluation which are the most important problems of this study. The ML models help software developers to accomplish their work in an easier and more reliable way with high code quality because the models are able to detect issues in a smart, effective, and scalable way with less effort to the developer.

Keywords: Bugs in Software, Software Defects, Machine Learning, Deep Learning, Software Code Analysis, Software Assurance: Reliability.

INTRODUCTION

In this world of evolving engineering, software bugs are perhaps some of the hardest challenges to overcome, as they can crash the entire system, create new security risks, and also enhance the software's maintenance cost. Due to the high cost of automation techniques based on ML, system failures and security flaws caused by everlasting software bugs has solidified the need for these automation techniques and surely needs to be addressed. Within the last decade researchers have been working on employing various ML methods to develop accurate defect detection automated systems that could alleviate the burden of software quality assurance processes. Efforts were initially focused on traditional supervised learning methods, where decision trees, random forests, and support vector machines were used to classify code segments as defective or non-defective based on certain pre-defined measures like cyclomatic complexity, lines of code, or past bug data. In due course, more advanced supervised methods were explored along with the introduction of deep learning paradigms based on convolutional neural networks, recurrent neural networks, and transformer models, which are more proficient at recognizing complex patterns within code that other methods fail to notice. Such models often benefit from scaling with larger data sets.

A major challenge is the availability and quality of the training data. Most machine learning models require enormous amounts of properly labeled data sets, which are not always present and does not always capture the conditions on the ground. The problem of data imbalance – where there is a lot more non-defective

code than defective code – makes it harder to train the model and usually leads to biased predictions and an increased incidence of false positives or negatives.

Objective

1. To study about automating a bug detector using machine learning.
2. Related work on automating a bug detector using machine learning.
3. Identified the research gap.
4. Finding and suggestions.

RELATED WORK

There are many studies about software bug detection using machine learning techniques. For example: This paper discusses the potential for change in machine learning-based bug classification, surveying recent developments, outlining existing challenges, and presenting future research avenues. Through the examination of important methodologies and their uses, this research points out how machine learning is transforming the discipline of software engineering, opening the door to more effective and trustworthy bug management systems (Vijay *et al.*, 2024).

This paper introduces a hybrid deep learning model to enhance software defect prediction. The method combines convolutional neural networks (CNNs) and long short-term memory (LSTM) networks to learn spatial and temporal relationships in software data. The authors experiment with the model on several datasets and show its better performance than traditional machine learning methods. With both sequence learning and feature representation, the model increases accuracy of defect detection in complex software

systems. The experimental findings confirm its scalability and reliability to operate across a variety of software environments. Limitations of the Paper: The paper is somewhat constrained with regards to real-world scalability and has a poor discussion on runtime performance as well as computation cost (Ahmed *et al.*, 2021).

This work suggests a novel approach to software bug classification based on a CNN-LSTM model. The CNN extracts spatial patterns in bug reports, and the LSTM extracts sequential dependencies, so the combined model is very effective in processing text data from bug repositories. Experimental results on open-source project datasets show that the model is very accurate in classifying bugs into various categories. The method seeks to address issues in bug prioritisation and triaging, ultimately leading to improved developer productivity and software reliability. Limitations of the Paper: The model's reliance on text data might limit its use in projects with noisy or missing bug descriptions (Md Nasim *et al.*, 2024).

This paper presents a systematic overview of different machine learning methods applied to software bug prediction. The overview categorizes current methods according to algorithm types, data preprocessing methods, and performance metrics. It identifies major trends in the area, such as the growing use of deep learning models and ensemble approaches. Limitations of the Paper: The research does not empirically verify its results and mainly consists of summarizing current literature (Tieng Wei Koh and Si Na Kew 2020).

This paper investigates the application of deep neural networks (DNNs) for software bug classification. The authors propose a DNN model that can handle both text and numerical inputs from bug reports. Through multimodal inputs, the model has greater classification accuracy in various software projects. The work entails extensive experimentation on open-source datasets, where the performance of the DNN is compared to conventional machine learning approaches. The findings indicate that the DNN not only enhances classification accuracy but also aids in automating bug triaging operations. Limitations of the Paper: Excessive computational demands and lack of interpretability restrict its real-world application in resource-scarce settings (Jyoti Prakash Mehra and Sourav Biswas 2023).

The paper proposes an ensemble approach to software defect classification where multiple machine learning models are employed together to boost predictive accuracy. The ensemble strategy leverages diversity of base models like decision trees, support vector machines, and neural networks for enhancing robustness. Limitations of the Paper: Increased computational complexity because more than one model is used and a narrow application scope in terms of real-time application scenarios (Issam H. Laradji and Mohammad Alshayeb 2021).

This paper suggests a hierarchical deep learning model to enhance the effectiveness of bug triaging in software projects. The model has a two-level architecture: a first-level classifier predicts bug types, and a second-level classifier assigns them to individual developers. The

method uses contextual features from bug reports, such as textual descriptions and historical information. Experimental tests on real-world data sets show that the model improves triaging accuracy and decreases resolution time, solving one of the primary challenges in collaborative software development. Limitations of the Paper: The approach could struggle with sparse data in projects with a small history of bug reports (Senthil *et al.*, 2018).

This paper suggests a hybrid machine learning model that integrates decision trees and gradient boosting to forecast software bugs. The model incorporates static code metrics and past defect data to detect potentially bug-prone code regions in codebases. Through the synergy of individual models, the hybrid model improves prediction accuracy. The authors test the approach on industrial-scale and open-source projects and show its scalability and efficacy in real-world applications. The approach also emphasizes interpretability, and hence it is appropriate for non-expert stakeholders. Limitations of the Paper: The reliance on static code metrics might restrict its use to dynamic or runtime-based faults (S. Delphine Immaculate, M. Farida Begam and M. Floramary 2019).

This work investigates the application of deep transfer learning methods to the automated classification of software defects. The authors adapt pretrained neural networks to domain-specific datasets with significant improvements in classification performance. The model is especially useful for projects with small labelled datasets, as it benefits from knowledge within similar tasks. The paper points out the scalability and versatility of the method across several domains and thus presents a viable solution for heterogeneous software environments.

Drawbacks of the Paper: The model's performance may decline in domains that differ significantly from the pretrained model's original training data.

This paper presents an attention-based recurrent neural network (RNN) model for classifying software bugs. The attention mechanism enables the model to focus on critical parts of bug reports, enhancing the interpretation of textual data. The approach is evaluated on open-source datasets, demonstrating its effectiveness in categorising bugs based on severity and type. The authors emphasise the advantages of incorporating domain-specific embeddings to improve the model's understanding of software terminology. The results indicate significant improvements over conventional RNNs and other machine learning models. Drawbacks of the Paper: The attention mechanism increases computational overhead, which may affect performance on large datasets (Prakash and Rao 2021).

This paper proposes an innovative machine learning-based approach for detecting and classifying software bugs. The technique integrates feature extraction from source code with textual analysis of bug reports to enhance classification accuracy. The model utilises gradient boosting algorithms to effectively handle imbalanced datasets and noisy features. The authors validate their method on open-source datasets, achieving competitive results in terms of accuracy,

precision, and recall. This study contributes to improving automated bug tracking and management workflows. Drawbacks of the Paper: The approach requires extensive preprocessing, which may complicate its implementation in dynamic development environments (Johnson and Thomas 2022).

This study investigates semi-supervised learning methods for enhancing bug detection in situations where labelled data is scarce. The suggested model utilizes labelled and unlabelled data in combination

through self-training and co-training approaches to improve its learning potential. Experimental outcomes show that the method remarkably enhances bug detection precision and decreases dependence on big labelled datasets. The research further presents semi-supervised learning's potential for low-resource projects as well as its applicability to large codebases. Drawbacks of the Paper: The model's performance may decline if the unlabelled data contains excessive noise or irrelevant information (Yang and Wu 2023).

Table 1: Comparative study of existing techniques.

Sr. No.	Title	Author	Technique used	Key Advantages	Drawbacks
1.	Software Bug Detection and Classification	Vijay <i>et al.</i> (2024)	Machine Learning	Identifies challenges and future research directions	Lacks empirical validation
2.	Software Defect Prediction based on Hybrid Deep Learning.	Ahmed <i>et al.</i> (2021)	Hybrid CNN-LSTM model	Exploit temporal dependencies for accurate defect prediction.	Limited real-world scalability, lacks runtime performance analysis
3.	Software bug classification based on a CNN-LSTM model.	Md Nasim <i>et al.</i> (2024)	CNN-LSTM architecture	Effective for bug triaging and identification accuracy.	Limited applicability for projects with noisy or incomplete bug reports
4.	A Systematic Review of Machine Learning Methods for Predicting Software Bugs.	Tieng Wei Koh, Si NA Kew (2020)	Machine learning techniques	Summarizes ML trends in bug detection.	No empirical verification of results, write mostly a summary of what others have done
5.	Deep Neural Networks for Software Project Bug Classification.	Jyoti Prakash Mehra and Sourav Biswas (2023)	Deep Neural Networks	High classification precision, supports multimodal inputs.	High computational cost, lacks interpretability
6.	An Ensemble Strategy for Software Defect Classification.	Issam H. Laradji and Mohammad Alshayeb (2021)	Decision tree, SVM, neural network.	Improve robustness, handles imbalanced data effectively.	Computational complexity, limited real-time applicability
7.	Hierarchical Deep Learning Model for Bug Triaging in Software Development.	Senthil <i>et al.</i> (2018)	Hierarchical deep learning	Enhances bug assignment accuracy, reduces resolution time.	sparse bug reporting history
8.	A Hybrid Machine Learning for Software Bug Prediction.	S. Delphine Immaculate, M. Farida Begam, M. Floramary (2019)	Decision trees+ gradient boosting	Scalable, interpretable for non-experts.	Relies on static code metrics, limited for dynamic effects

RESEARCH GAP

This literature review presents comprehensive studies in the field of automating a bug detection & identification. This broad literature review serves as a roadmap to uncover the limitations of current approaches and leads to the improvement of well-defined research problems. Some initial progress has been made toward automating bug detection, but there are still some pertinent research gaps. One of the aspects is scalability, deployment and execution; the models that work well in very controlled environments do not perform well in large-scale real-world software projects due to their execution costing a lot of resources. Another aspect is class imbalance together with under labeled data sets; this occurs when the available defective code is hugely less than the non-defective code, resulting in skewed predictions and aggravated false positive or negative ratios. To add to this, a tangible barrier is lack of interpretability and

explainability because most deep learning models are known as "black boxes" and their reasoning is completely absent. Another impediment to progress is domain adaptation and generalization problems since a given type of software tends to have models built over it and those models are poor performers in other domains. Moreover, the level of integration of ML-based bug identification into the Software Development Life Cycle (SDLC) is very minimal due to absence of research on active bug identification, integration with the CI/CD pipeline, and automation of debugging grails. This is yet another gap because there has been a lack of attention on the self-supervised and unsupervised learning which seeks to minimize the use of labeled data. Furthermore, the optimization of the computation and efficiency of the deep learning model poses a problem in that they are too costly in resources, making them impractical for real-time use. Besides, the ambiguity is a valid problem for noisy and incomplete

bug reports because most bug tracking systems are populated with loose and disorderly claims. Moreover, the estimate based on comparison is largely overlooked which softens the boundary at which bug detection using machine learning surpasses that of rule-based systems. Finally, more attention needs to be directed to the security and ethical considerations of implementation of adversarial attacks on ML models and bug classification biases in the ML system for the unsupervised defect detection methodology for the ML system. Addressing this gap would enhance the effectiveness, validity, and practicality of offensive ML-enabled bug detection systems.

FINDING AND SUGGESTIONS

This review discusses the problems and challenges associated with using machine learning to detect the bug in software projects. The main challenges include setting models, parameters correctly and avoiding overfitting. Machine learning is highly efficient to use for automating a bug detector and allows accurate identification of software defects while at the same time cutting down on the time spent on debugging. There are various ML techniques through which bug detection and classification can be performed, such as deep learning models (CNN, LSTM, DNN), hybrid models and ensemble methods. However, high computation costs, lack of transparency, scalability in society, data imbalance and other such issues are hindrances that are too large to be crossed. Much of the current models are too reliant on structured and labeled data sets that diminish their utility for dynamic real time software development. Also, most of the techniques cover the problem classification without considering the important detection and prevention stages of a problem for the most probable software vulnerabilities.

These recommendations aim to enhance the effectiveness of automated bug detection through the use of machine learning (ML) by concentrating on issues such as data availability, interpretability, diagnostic skills, and the practical problem of demographic imbalance in class representation in the dataset. To increase the precision of categorization with an ML approach, it is possible to create hybrid models that blend classical and deep learning. Also, transfer learning is beneficial for alleviating the low data availability problem since it enables easier transition of software models into different scope environments. For the purpose of improving real-time bug detection, low-complexity models for edge devices should be considered as they decrease the work needed to process the information and increase the speed of the system. Finally, automating bug triaging and prioritization with reinforcement learning can improve the efficiency of debugging by ensuring that key issues are resolved in a timely manner which makes triaging more efficient. Implementing these recommendations will increase the scalability, interpretability, and practicality of automated ML bug detection systems.

CONCLUSIONS

Integrating machine learning (ML) algorithms to bug tracking and detection improves the quality and

dependability of software. Includes manual analysis, code inspection, and static processes are common approaches to bug detection, but are time consuming and prone to human error. The use of machine learning models increases bug detection accuracy and decrease false positive rates through patterns identification, anomalies recognition, and advanced deep learning capture of complex software bugs. Self-Supervised models, as well as advanced deep learning and optimization tools, show greater potential in more accurately detecting software bugs. These models help predict software threats through gauging class instances of bugs in historical data along with offering plausible solutions. This approach gives rise to software feature search automation post bug fixing, significantly cutting down time spent on debugging. Additionally, machine learning benefits from feature selection and dimensionality reduction during the automation development phase of systems to improve model performance. There are many untapped areas with the model training needing manual working intervention such as the so called the quality unbalance ratio and constant model tuning to fit the new dynamics of software frameworks.

FUTURE SCOPE

The prospect of machine learning systems in automated bug detection is hopeful, with AI, deep learning, and automation set to change the field of software quality assurance. A primary area of concern is the seamless integration of machine learning within DevOps processes and CI/CD pipelines, which allows for automatic bug detection and resolution within software deployment. Hybrid and ensemble deep learning models with other classical and reinforcement machine learning approaches will enable greater flexibility and precision. Additionally, self-supervised and transfer methods will foster greater multi-language defect localization with less reliance on exhaustive labeled data sets. The emergence of explanatory AI provides greater insight into how ML-powered bug detectors arrive at their conclusions, which is vital to trusting them. Other expected changes will focus on self-healing where models not only locate bugs but also generate and execute fixes. This will significantly shorten the period spent on manual debugging of programs. In addition, future models will go beyond cross-language bug detection to include multi-platform support for more comprehensive universal software bug fixing tools. The collaboration between humans and Artificial Intelligence when it comes to debugging will reach a point where AI tools assist programmers, and algorithms can learn from their inputs to improve. The aptitude of automated bug detection will attain new levels and software development utilizing machine learning and AI will improve efficiency, accuracy, precision, and scalability. In turn, there will be more immediacy in the need for faster development cycles, enhanced security to modern applications, and reduced costs.

Acknowledgement. It is the sincere wish of the author to thank all the members of faculty of department for

their rich guidance, unflinching support, and constant encouragement throughout the course of this research work.

REFERENCES

- Ahmed Bahaa Farid, Enas Mohamed Fathy (2020). Software defect prediction via hybrid deep learning model. *IEEE Trans Software Eng.*
- Issam H. Laradji and Mohammad Alshayeb (2021). An ensemble approach for software defect classification. *J Syst Softw.*
- Johnson, M. and Thomas, B. (2022). A novel bug detection and classification technique using ML algorithms. *Soft Comput.*
- Jyoti Prakash Mehra and Sourav Biswas (2020). Deep neural networks for bug classification in software projects. *Empir Softw Eng.*
- Md Nasim Uddin Ansari and Pankaj Richhariya (2020). Bug classification using CNN-LSTM in open-source software systems. *Comput Sci Rev.*
- Prakash, J. and Rao, A. S. (2021). Classifying software bugs using attention-based recurrent networks. *Softw Qual J.*
- Senthil Mani, Anush Sankaran, and Rahul Aralikatte (2021) Hierarchical deep learning model for bug triaging in software development. *Autom Softw Eng.*
- S. Delphine Immaculate, M. Farida Begam, M. Floramary (2021) A hybrid machine learning model for software bug prediction. *Softw Pract Exp.*
- Tieng Wei Koh, Si NA Kew (2020) A systematic review of machine learning techniques for software bug prediction. *Appl Soft Comput.*
- Vijay Dhanaraj Sonawane, Pratap Singh Patwal and Vinod S. Wadne (2024). Software Bug Detection and Classification Using Machine Learning Techniques.
- Yang, H. and Wu, J. (2023) Enhancing bug detection with semi-supervised learning. *Softw Syst Model.*