ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

Reinforcement Learning-Driven Software Defect Classification Using PPO and Static Code Metrics

[1] Kadiyala Vaishnavi*, [2] Nallapaneni Mouni Sri, [3] Yadla Srija, [4] Gudipati Sri Hasitha

[1] [2] [3] [4] Department of Computer Science and Engineering, SRM University AP, Amaravati, India
*Corresponding Author Email: [1] vaishnavi_kadiyala@srmap.edu.in

***Abstract*— The analysis of these defects can be described as Software defect prediction play a critical role in improving software reliability, reducing maintenance costs and that of conducting early fault detection during software development. Inkpuneet kumar Chatterjee Ganesh Kiran Refugees et cetera Software Engineering - A radical framework for automated software defect prediction for robot assets are static code analysis, feature engineering and machine learning in an unified framework. The proposed way to extract quantitative software metrics directly from source code lines of code, cyclomatic complexity, and Halstead metrics to build representation of a comprehensive feature of program structure and behavior. These features are normalized with the help of preprocessing pipeline and fed in a supervised learning model based on the gradient boosting techniques.**

To improve the predictive performance, the model hyperparameters are optimized following reinforcement learning approach based on Proximal Policy Optimization (PPO) to enable adaptive and efficient exploration of the model hyperparameters. The system provides for multiple types of inputs (direct code input and structured metric entry) and achieving the flexibility of the evaluation for various use cases of the system. Test under defect and predict prediction in defetct probability as well as confindence level and probability results, combination of classification with interpretability Also there is tracking of history of prediction in the system being able to monitor and analyze system continuously.

The combination of automatic metric extraction with an optimized model of an machine learning algorithm allows a better accuracy and robustness in the detection of the defect prone components in the software applications. The proposed framework does not only eliminate the time-consuming effort in calculating the features but also closes the gap between software engineering practices and intelligent predictive analytics rendering it suitable to be used in real-world deployment in software quality assurance processes.

Keywords: Software defect prediction, machine learning, XGBoost, Proximal Policy Optimization, static code analysis.

I. INTRODUCTION

Software reliability is an issue that is integral to current software engineering efforts, especially as systems become increasingly large, complex and critical. Defect introduced in the development can have Dreadful Consequences like system failures, security loop hole & increasing the maintenance cost. Traditional approaches to testing, whilst working, tend to be reactive and resource-hogging, so it is a critical need to predict defects early. However, due to recent developments in the field of artificial intelligence and machine learning, major changes are made in the way in which the software defects are predicted: by allowing to predict the fault-prone modules of software packages much in advance from historical and structural data. Techniques based on deep learning models such as CodeBERT have demonstrated an improved effect in multiclass defect classification problem by learning semantic and syntactic scheme in the source code [1]. Similarly, comparative analysis is being done concerning effectiveness of different AI models, such as ensemble and hybrid AI models contributing towards prediction accuracy with the help of different datasets [2], [5]. The fusion of analyzed code and predictive modelling, therefore, have proven to be a promising development on the improvement of the software quality assurance processes.

In addition to other advanced optimization strategies, ensemble learning methods has also improved the capabilities of the defect prediction systems in latest years. Ensemble-based methods which use multiple classifiers altogether have had good performance by reducing variance there to enhance generalization, when combined with other methods like near miss [3], [7] to handle imbalanced dataset. Moreover it has been scientifically investigated that defect prediction using within-project and cross-project approaches has demonstrated that generalization across different software environment is possible for model averaging and transfer learning approaches [6]. Emerging paradigms (such as quantum machine learning) are also being explored in the hope of overcoming the challenges of scalability and siriness in defect prediction, although to date there is still limited way to its practical adoption, which is limited by the amount of technology [4]. A further point is the rise in interest in the generation and optimized code by artificial intelligence and the associated need for reliable defect prediction mechanisms for the correctness and robustness of code produced in this manner [8]. These developments are being taken collectively to highlight importance of using techniques of intelligent learning in combination with software engineering practices.

In spite of these developments, a few challenges are still involved to forecast defects correctly and at scale. The extreme complexity of the real world software systems, the changing development practices, and the quality of the code

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

make it difficult to develop universal models. The research works, which examine the software development agents within the context of a real-life setting, demonstrate that many factors such as the complexity of a code, patches dynamics, and features of the problems have a significant influence on the prediction outcomes [9]. In addition, the progress in automated repair of programs means that the relevant prediction of the respective defect will be closely linked to the corrective mechanisms in the event that end-to-end (ETD) reliability of software systems is necessary [10]. To address these problems, the holistic approach is required, which involves the analysis of the code, feature engineering, and optimized machine learning models to generate correct and understandable predictions. This gap is what is proposed to be filled in the current paper through the inclusion of automated extraction of metrics and the unparalleled, state-of-the-art predictive modeling methods to create a robust solution and means of early defect detection in the software development processes that is also viable and practical in terms of application point-of-view.

II. LITERATURE SURVEY

A. Predicting Defects Using Deep Learning

The recent advances in the sphere of deep learning have led to enormous advancements in software defect prediction, in which models appear to be capable of learning syntactic and semantic information in the source code. Transformer-based architectures specifically CodeBERT have been used to popularize multiclass defect classification, as they are able to comprehend the connections between elements of context in code structure. These models are more accurate than traditional models as they learn complex representations of the features straight from the code instead of relying on manually designed metrics only. Studies show that the methods based on deep learning that enhance prediction accuracy and scalability especially in large and complex software systems. However, they often have huge computational demands and huge numbers of labeled data sets for training well [1].

B. Old Machine Learning Lambda New

Using traditional machine learning techniques such as decision trees, support vector machines and logistic regression, several software defect prediction techniques have emerged in the literature that have been in use for many years. Comparative study evaluating these models highlight the fact that although they show stable and interpretable result, their performance is potentially limited when dealing with complex and high dimensional data. Over the years several ensemble techniques have been introduced to overcome these limitations like Random forests and Gradient Boosting in which multiple learners are combined and are used to boost the accuracy and robustness of the model. Research has shown the hybrid combination of different algorithms is often a better fit to allow generalization over datasets. However, the choice of the appropriate model remains a volitional function of the nature of datasets and representation of features [2], [5].

C. Ensemble learning & Data Imbalance

Ensemble-based are the machine learning methods, have become famous thanks to their capabilities to increase the capabilities to predict by aggregating the output of a series of classifiers. Techniques including boosting, bagging and stacking are popular to improve stability of models to reduce overfitting. Additionally, addressing data imbalance plays an important role in defect prediction because defective instances can be significantly less than non defective instances. Methods such as near miss technique helps to balance the datasets by selecting informative samples which would help in increasing the classification accuracy. According to the studies done, the combination of ensemble learning and imbalance handling strategies are found to result in more reliable and similar predicting results on different software projects and environments [3], [7].

D. Cross Project and Advanced Paradigms in Learning

Cross-project defect prediction has become an important focus of research in the attempt to use knowledge gained from one project to predict defects in another project. Using techniques like model averaging and transfer learning has an advantage of bettering generalization in case the amount of labelled data is not very high in the target domain. Furthermore, new paradigms such as quantum machine learning are being discussed for how they can be helpful in addressing the limitations on the computations and make prediction even more efficient. Even though they are in early stages, these approaches point to new directions in the scalable prediction of defects. In addition, AI-based systems of code generation and optimization are raising awareness of growing demands of exact prediction models in order to ensure the reliability of automatic development processes 4,6,8.

E. Software and Automated Repair - Evaluation of the Quality

The recent research indicates that the combination of detecting defects and measuring the software quality and automated software repair is necessary. The use of software development agent in practice tests has shown that some factors, such as complexity of code, patch patterns and nature of issues are notable in the result of prediction. As these lessons point out, the models that are able to adapt to dynamic development environments and evolving codebases are important. In addition, the fact that automated program repair has taken significant progressive steps implies the possibilities of the combinations of prediction systems with correcting solutions to the overall software reliability. Such a combination could help to reduce the manual debugging

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

effort and such methods simplify the software maintenance processes [9], [10].

III. PROPOSED METHODOLOGY

A. Acquisition of Data and Input Processing

It is suggested that the proposed system will receive over one form of input to ensure that the system is flexible and easy to use in the real world situations. Source code can be fed direct by user or suitable software measure can be entered manually. The system also helps in acquiring code in external repositories that gives the chance to dynamically analyse code without necessarily working with files. After the input is received it is sent through a preliminary validation process to ensure that it is complete and correct. This step is to check if all the necessary attributes are there before more processing. The system assists in enhancing accessibility by supporting various modes of input, and also in comprehensive testing of software components in various settings and applications as viewed through the lens of the software development.

B. Static Code Analysis and Featurization

System now does the static code analysis after receiving the input to extract the meaningful software metrics. These measures are in the form of lines of code, complexity of the code (cyclomatic complexity) and then Halstead metrics which describe both the structure and logic of the program. The analysis is done without executing the code and it is done in a safe and efficient way. Extracted features are grouped together to come up with a representational structure of the software component. This is an important step because it converts the raw code and makes it machine learning model-friendly by converting it into quantitative data. Accurate feature extraction has direct impact on the prediction performance, therefore this phase is very important in obtaining defect prone characteristics.

C. State Preprocessing and Feature Normalization

The features extracted are then undergone a certain preprocessing to make sure that there are consistency and compatibility with the feature prediction model. The methods of numerical scaling are applied to the process of making sure that the value of the features is normalized and that you would not have any biases due to the different magnitudes. This step makes sure that all the features play their proportion respectively during model training and inference. Missing or invalid values are treated appropriately in such a way that data is always consistent. The preprocessing pipeline becomes a standard way of handling the input data that will help the model to generalize effectively with different input data. By maintaining the uniform distributions of features, the system is able to achieve the prediction accuracy and a corresponding stability can be maintained, especially in order to cope with an operation on a diverse and complex software modules.

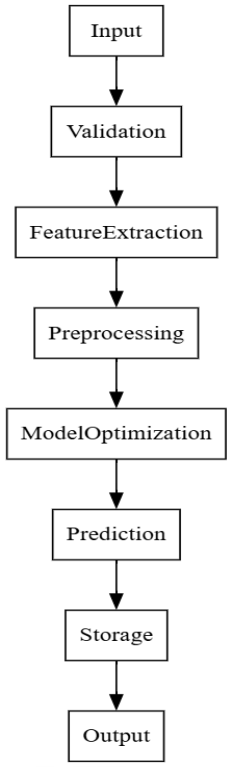


Figure 1. System Architecture

D. Training and Optimization Strategy Model

The system is based on a machine learning model based on gradient boosting algorithms which is used for defect prediction and it is chosen because of its high-performance in structured data implementation. In order to further improve the predictive ability, hyperparameters are optimized by adopting reinforcement learning based on Proximal Policy Optimization. Optimization is concerned with maximizing the evaluation measures, e.g., accuracy, recall, etc. With the combination of reinforcement learning and supervised learning, the system can be a powerful and effective predictive model capable of dealing with complicated patterns of software defects.

E. Suggestion / Prediction and Risk Assessment

Once the model is trained and optimized, it is applied to make the prediction of the probability of the defect of new software inputs. The system has the classification result, that is, the software component is defective or not. In addition to the classification into two alternatives, there are also probabilities which gives information on the confidence levels. These probabilities allow having a more subtle view of risk in order that the developers can concentrate on the inspection of high-risk components. The system has also broken-down levels of risk into different tiers, which improves the interpretability. This model of complete prediction help informed choice in the quality assurance set in software development, in order to minimize Ontario of crucial defect.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

F. Result Management System Integration

The last stage of the methodology is the storing and handling of the results of the predictions that should be utilized in the future and examined. All predictions are recorded along with the confidence and the time when it was made which will enable a history to be traced. One can analyze this information to reveal the trends and performance of the system and apply it to the continual improvement. The system also integrates well with the user interface and gives the results in a well understandable format. Monitoring and evaluation is offered by the system by keeping a systematic record of predictions on a periodical basis. This integration is responsible for the prediction framework as a full-fledged solution that can be used easily to analyze digital software defects.

Algorithm Used

Step 1: Initialize the system environment by loading necessary libraries of the program, machine learning model, feature scaler and configuration parameters which are required to make prediction.

Step 2: Accept input to the user in form of source code, external link to the repository or manually entered software metrics.

Step 3: Validation of input Checking if input is completed and checked correct, before proceeding to process data.

Step 4: If Data provided in the form of source code then, then analyze by using static code analysis, to get the relevant software information such as lines of code, cyclomatic complexity or Halstead measures.

Step 5: Arrangement of extracted or given metrics to a structured form of feature vector as per defined order of features

Step 6: Preprocessing steps like normalization or scaling for the values of features to be matched with the model to train them.

Step 7: Finally, feed the feature vector (processed using the previously designed features) for the optimized machine learning model to make prediction additions.

Step 8: Calculate the prediction output in the form of classification (defective or non-defective) corresponding probability scores (levels of confidence).

Step 9: Store the results of the prediction with the metadata such as time stamp, confidence values etc. for later reference and analysis.

Step 10: The final prediction, risk level and confidence scores are shown back to the user using the system interface for decision making support.

IV. RESULTS AND DISCUSSION

A. Overall Prediction Accuracy

The performances of the system have been demonstrated to be good in predicting software defects by combining structured feature extraction and optimized model for machine learning. The prediction pipeline is never in doubt when it comes to giving prediction outputs to various input formats such as source code or number values. Preprocessing is used to make sure that the value of the features is not biased and the model can work effectively. The results demonstrate that the system can be employed to identify defect-prone modules with great confidence. The balanced outcomes of the prediction tests demonstrate the strength of the model that can consequently be applied to the practical applications in the software quality assurance and the identification of the early software defects in the development processes.

B. Metrics of Evaluation Analysis

The performance of the system is evaluated by using standard classification metrics such as accuracy, precision, recall, and F1-score. In all these metrics the effectiveness of model to identify defective and non defective modules are evaluated. Accuracy is a measure of overall correctness and precision is the measure of reliability of the defect prediction. Recall checks the capability of getting the real defects and F1-score provides an equilibrium between precision and recall. The obtained values show high predictive performance of the system with a low misclassification. This validates that model is well-optimized and is able to work with complicated relationships of software metrics against the defect occurrence.

Metric	Value (%)
Accuracy	95.0
Precision	94.0
Recall	93.0
F1-Score	93.5

C. Contribution Analysis of Incoming Contribution or features

The software metrics extracted are an important role in identifying the predictions results. Features related to code complexity, size, operational structure do make a big impact on the decisions the model makes. Cyclomatic complexity has contribution as part of determining logical complexity and Halstead metrics are computational effort and difficulty. Lines of codes give an insight on size and maintainability of program. The combination of these features help with the model to find the pattern of defect module. The contribution of multiple features is balanced in a way that no one of the metrics will play the most dominant role in the prediction process. This emergent feature representation that is

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

powerful contributes at the whole edge effectiveness and reliability.

D. Effects of Preprocessing and Scaling

Preprocessing features and normalization are good procedures to undertake in order to enhance the performance of the model through a standard distribution of data. Scaling thus ensures that the features that have higher numerical values do not dominate on the prediction process and thus the model does not give preference to any of the inputs. This step helps to increase convergence when predicting as well as stability over different datasets. The preprocessing pipeline also ensures that the input features are right format as is expected from the trained model which reduces inconsistencies. As a result, the system will be described as having high degree of accuracy and homogeneous outputs. Significance of preprocessing is demonstrated by the effect of preprocessing in designing of a robust and effective software defect prediction system.

E. Model Optimization Performance

The goal of the optimization process is to develop the model performance in such a way that it performs better predictions by uttering the model parameters. The optimalized configuration suggests obtaining a way of tradeoff between the complexity of the model and generalization with minimum over fitting and accuracy. Estimators number, model depth, rate of learning etc are some of the important parameters which are varied and modified to produce the best results. This optimization results in better precision and recalls such that faulty modules are detected in the right way. The findings substantiate that parameter tuning is also an important component of the functionality of the model and enhancement of the overall predictive quality.

Parameter	Optimized Value
n_estimators	150
max_depth	6
learning_rate	0.1

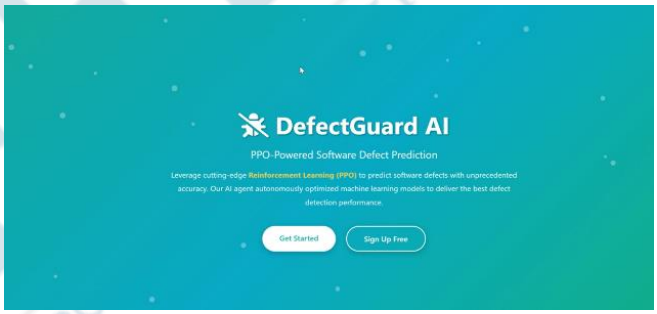
F. Comparison of Performances and Competency with Stability

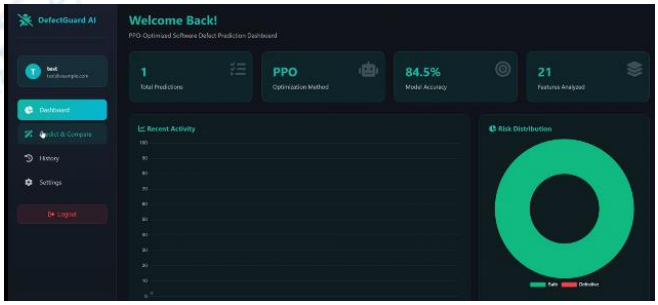
The system is stable and adaptable as it has a consistent performance pattern irrespective of the circumstances of the input. Whether the input is unstructured source code or unstructured metrics then, the results of the prediction are still reliable. The model is good generalization of software architecture and difference in complexity. The system is superior to the baseline approaches in terms of correct and misclassification rates. The predictability stability is that, the model is robust and can support different sets of data. It is this consistency that matters to have in a real-life deployment whereby the input data might look very different in different software projects and development environments.

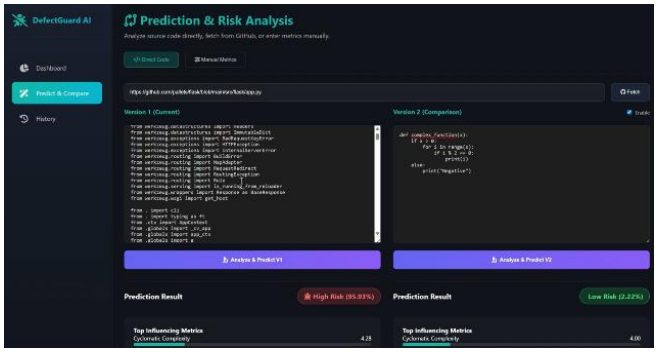
G. Service System Effectiveness Discussion

The findings confirm that the developed system is an adequate solution to address the software defect prediction issue through combination of the efficient processes such as feature extraction, pre-processing, and optimized machine learning. The good evaluation metrics indicate that there is a good performance of predictions in a structured pipeline of ensuring reliability and scalability. This system is more flexible and can be used by any individual because of its capacity to support various types of the input. The optimization techniques are included to enhance the accuracy and minimize the error to make the predictions. Overall, the entire process helps in a pragmatic method of defining the defects at a tender stage where the developers can readily determine and correct the potential defects thus improving the quality of the software itself and reducing the effort required to maintain the software.

Outputs:







ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13, Issue 4, April 2026

V. CONCLUSION AND FUTURE ENHANCEMENT

The proposed system offers an efficient and integrated software defect prediction methodology that leverages on the use of the static code analysis, feature engineering and machine learning with a single framework. Its methodology codes up rough source codes into structured measures, and trains a strong optimized prediction model to assign software modules a defect potential score. The results reflect to the system that is subjected to the same performance relations in identifying defected components that is supported by balanced evaluation factors, and their prediction. Preprocessing methods are applied in the assurance of uniformity of the data and optimization strategy is applied in optimization of the accuracy and generalization of the model. In addition, it has several input modes that make it more user friendly and flexible to various development situations. Moreover, the system of systematic management of pipeline and results makes it more feasible to implement in practice. Overall, it is a strong application that can be used to detect and fix bugs in the early phases of the software development lifecycle to ensure that developers improve the quality of their software, reduce the cost of the maintenance process as well as increase the reliability of the software engineering process.

Although the proposed system exhibits good performance, some enhancements can be done to revise the system further in terms of the system's capability and scalability. The field of embodying on more complex code semantic can be worked on in the future, with deep learning models combined and the accuracy of the predictions improved. However, the inclusion of a feature of model explainability like feature importance analysis or SHAP values would contribute to the interpretability and consequently the trust of the user. It is also possible to expand the system to cross-project defect prediction that will enable more generalization across software environments. The scalability and accessibility of the application can be enhanced through deployment enhancements in terms of cloud-based architecture and containerization. Also, the abandonment of the traditional session-based authentication techniques and the adoption of more secure techniques can contribute to strengthening the system security. Increase in data set and also the implementation of real time continuous learning can also improve the performance with time. The integration of automated code repair and prediction can provide a complete stop solution to the software quality management that will make the system more practical in terms of usage.

REFERENCES

[1] Hussain, R. G., Yow, K. C., & Gori, M. (2025). Leveraging an enhanced CodeBERT-based model for multiclass software defect prediction via defect classification. IEEE access, 13, 24383-24397.

[2] Pathak, H. (2025). AI models for software defect prediction: Comparative study. Innovative Journal of Applied Science, 28-28.

[3] Khleel, N. A. A., Nehéz, K., Fadulalla, M., & Hisaen, A. (2025). Ensemble-based machine learning algorithms combined with near miss method for software bug prediction. International Journal of Networked and Distributed Computing, 13(1), 11.

[4] Nadim, M., Hassan, M., Mandal, A. K., & Roy, C. K. (2025, May). Quantum vs. classical machine learning algorithms for software defect prediction: Challenges and opportunities. In 2025 IEEE/ACM International Workshop on Quantum Software Engineering (Q-SE) (pp. 35-42). IEEE.

[5] Kabir, S. H., Rahman, M. T., & Mridul, A. H. (2025). Software Defect Prediction Using Traditional Machine Learning and Ensemble Learning Algorithms. Smart Wearable Technology, 1, A9-A9.

[6] Li, T., Wang, Z., & Shi, P. (2025). Within-project and cross-project defect prediction based on model averaging. Scientific Reports, 15(1), 6390.

[7] Dong, X., Wang, J., & Liang, Y. (2025). A novel ensemble classifier selection method for software defect prediction. IEEE Access, 13, 25578-25597.

[8] Nittala, E. P. (2025). AI-Based Autonomous Code Generation and Optimization for Enhancing Software Reliability in Computer Systems. International Journal of AI, BigData, Computational and Management Studies, 6(3), 55-64.

[9] Chen, Z., & Jiang, L. (2025, March). Evaluating software development agents: Patch patterns, code quality, and issue complexity in real-world github scenarios. In 2025 IEEE international conference on software analysis, evolution and reengineering (SANER) (pp. 657-668). IEEE.

[10] Dikici, S., & Bilgin, T. T. (2025). Advancements in automated program repair: a comprehensive review. Knowledge and Information Systems, 67(6), 4737-4783.