ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

AI-Assisted Python Code Quality and Bug Risk Analysis Platform

[1] Srilakshmi S, [2] Vandhana V, [3] Nithyasree R, [4] Manisha N, [5] Pamila A
Student , Dept. of Computer Science and Engineering, Er. Perumal Manimekalai College of Engineering
Hosur, Tamilnadu, India.
deekshud881@gmail.com, vandhana713@gmail.com, nithyasree2233@gmail.com,
manishahgh1234@gmail.com, pamilacharles1313@gmail.com

Abstract: ***An important part of current software development is software quality assurance, especially for dynamically written languages like Python, where undetected bugs can result in serious system failures. In order to detect and categorize bugs in software, this paper introduces an AI-assisted Python code quality and bug risk analysis platform that combines machine learning with static code analysis. The proposed approach uses a collection of 20,000 labelled samples representing 100 different categories of bugs to evaluate Python source code. Code characteristics are represented by extracting static code metrics like Lines of Code (LOC), cyclomatic complexity, and maintainability index. A Random Forest classifier for bug prediction is trained using these features. The model's accuracy of almost 92% shows how well it can spot possible bugs. The solution improves developer insight and efficiency by offering severity assessment, thorough explanations, and recommended fixes for bugs found in addition to classification. An interactive and user-friendly interface is made feasible by the platform's design, which uses a web-based frontend created using HTML, CSS, and JavaScript and a Flask-based backend. According to the results of experiments, the suggested method improves the efficiency of bug detection and supports proactive code quality control.***

Keywords - Python Code Analysis, Bug Prediction, Machine Learning, Random Forest, Static Code Analysis, Software Quality, Cyclomatic Complexity, Maintainability Index, Flask

I. INTRODUCTION

The discovery and prevention of bugs is a difficult task in the software development lifecycle due to the increasing complexity of software systems. To guarantee reliability, maintainability, and general software quality, early detection of bugs is crucial. Especially in large-scale projects, manual code review and traditional debugging methods are frequently time-consuming and subject to human error. Python is a popular high-level programming language that is often used in fields like artificial intelligence, data science, and web development. Its dynamic nature, however, may result in hidden bugs and runtime errors that are challenging to find during development. This demands the use of intelligent and automated techniques that can analyse the quality of code and predict possible bugs. Recent advancements in machine learning have enabled the development of predictive models that can identify bug-prone code segments based on historical data and code metrics. Static code analysis techniques further enhance this process by extracting meaningful features such as lines of code, cyclomatic complexity, and maintainability index, which provide insights into code structure and quality. In this research, we propose an AI-assisted Python code quality and bug risk analysis platform that predicts bug types and evaluates their severity by combining machine learning and static analysis. A Random Forest classifier that was trained on a dataset of 20,000 labelled samples covering 100 different bug categories is used by the system. The software helps developers improve the quality of their code by not only detecting potential bugs but also offering explanations and recommended fixes.

II. LITERATURE SURVEY

Software defect prediction has become an important research area in software engineering, aiming to improve software quality by identifying fault-prone modules early in the development lifecycle. Traditional approaches relied heavily on manual code reviews and static analysis tools, which are often time-consuming and limited in scalability. With the increasing complexity of software systems, automated techniques have been widely explored to enhance defect detection efficiency.

Recent studies have demonstrated the effectiveness of machine learning techniques in software defect prediction. Various algorithms such as Support Vector Machine (SVM), Naïve Bayes, Decision Trees, and Artificial Neural Networks have been applied to classify defect-prone code using historical data and software metrics (). These approaches leverage features such as lines of code, cyclomatic complexity, and other structural metrics to build predictive models. Research has shown that

machine learning-based methods significantly improve defect detection compared to traditional techniques by enabling early identification of potential faults ().

Among different machine learning models, ensemble learning techniques, particularly Random Forest, have gained significant attention due to their robustness and high prediction accuracy. Studies indicate that Random Forest models outperform several baseline algorithms by effectively handling high-dimensional data and reducing overfitting (). Additionally, empirical analyses have reported that optimized Random Forest and ensemble approaches can achieve very high accuracy levels, making them suitable for practical defect prediction systems ().

Furthermore, recent research has explored advanced approaches combining static code metrics with machine learning and deep learning techniques. These studies highlight the importance of using diverse datasets and multiple evaluation metrics such as precision, recall, and F1-score to ensure reliable performance (). Ensemble and hybrid models have shown promising results, particularly in handling imbalanced datasets and improving prediction reliability.

Despite these advancements, existing systems face several limitations. Many approaches focus only on defect prediction without providing explanations or actionable insights for developers. Additionally, the lack of severity assessment and automated fix suggestions limits their practical usability in real-world development environments. Moreover, no single model performs optimally across all datasets, highlighting the need for more adaptive and intelligent systems ().

To address these limitations, this paper proposes an AI-assisted Python code quality and bug risk analysis platform that integrates static analysis with a Random Forest-based prediction model. Unlike existing approaches, the proposed system not only predicts bug types but also provides severity analysis, explanations, and suggested fixes, thereby enhancing developer productivity and supporting proactive software quality management.

III. METHODOLOGY

The proposed system follows a systematic methodology that includes data preprocessing, static feature extraction, model training, and bug classification. Initially, the input data, which consists of Python source code files, is collected and processed through a web-based interface. During preprocessing, unnecessary characters, comments, and formatting inconsistencies are removed to ensure clean and standardized input for analysis.

Feature extraction is performed using static code analysis techniques, where important software metrics such as Lines of Code (LOC), cyclomatic complexity, maintainability index, number of functions, and warning counts are computed. These features represent the structural and logical properties of the source code and are converted into numerical form for machine learning processing.

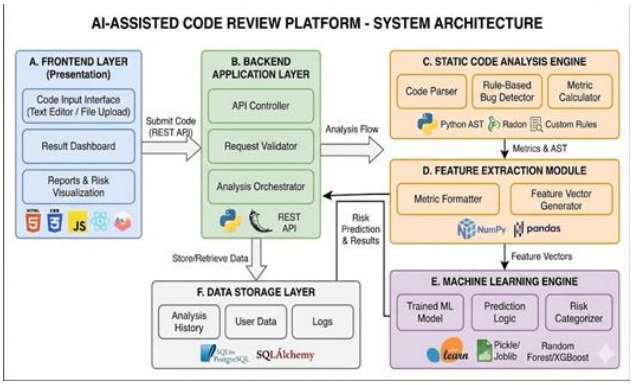

Fig. 1: System Architecture of the AI-Assisted Bug Detection Platform

The extracted features are then passed to the Random Forest classifier, which is an ensemble learning method that constructs multiple decision trees and combines their outputs to improve prediction accuracy. The prediction of the model is based on majority voting and can be expressed as:

Final Prediction = mode ($T_1(x)$, $T_2(x)$, ..., $T_n(x)$)

Where:

- T_1, T_2, ..., T_n represent individual decision trees

- x represents the input feature vector

The model is trained on a dataset of 20,000 labelled samples covering 100 different bug types. During training, the algorithm learns patterns associated with different bug categories and their corresponding feature distributions.

After classification, the system evaluates the severity of the predicted bug by assigning a risk level such as low, medium, or high based on predefined thresholds and model confidence. Additionally, an explanation module is integrated to provide meaningful insights into the detected issue along with suggested fixes.

Finally, the system outputs the predicted bug type, severity level, explanation, and recommended solution through the user interface, enabling developers to identify and resolve issues efficiently. This methodology ensures an automated and intelligent approach to software defect detection and code quality improvement.

IV. IMPLEMENTATION

The proposed AI-assisted Python Code Quality and Bug Risk Analysis Platform is implemented using a combination of web technologies, static analysis tools, and machine learning techniques. The system is developed with a modular architecture consisting of frontend, backend, feature extraction, and prediction components.

The frontend of the system is developed using HTML, CSS, and JavaScript, providing an interactive interface for users to input Python source code either by pasting or uploading files. The interface is designed to display analysis results in a clear and user-friendly manner. Communication between the frontend and backend is established using HTTP requests.

The backend is implemented using the Flask framework in Python, which handles request processing, input validation, and integration with the machine learning model. Upon receiving the input code, the backend performs preprocessing steps such as cleaning, formatting, and preparing the code for analysis.

Static code analysis is carried out using Python-based analysis libraries such as Radon, Pylint, which is used to extract important software metrics including Lines of Code (LOC), cyclomatic complexity, and maintainability index. In addition to library-based analysis, custom scripts are implemented to compute additional features such as function count and warning indicators. These extracted metrics are organized into a structured feature vector suitable for machine learning input.

The machine learning component is developed using the Scikit-learn library, where a Random Forest classifier is trained on a dataset of 20,000 labelled samples representing 100 different bug types. The trained model is integrated into the Flask backend to perform real-time predictions. Given an input feature vector, the model predicts the bug type and provides a confidence score based on the learned patterns.

Table I presents the tools and technologies used in the implementation of the proposed system.

Component	Technology Used
Frontend	HTML, CSS, JavaScript
Backend	Flask (Python)
Static Analysis	Radon and Pylint Library
Machine Learning	Random Forest (Scikit-learn)
Dataset	20,000 Labelled Samples
Feature Extraction	LOC, Complexity, MI

ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

Furthermore, a result interpretation module is incorporated to evaluate the severity level of the detected bug and generate meaningful explanations along with suggested fixes. These outputs are processed and sent back to the frontend for visualization.

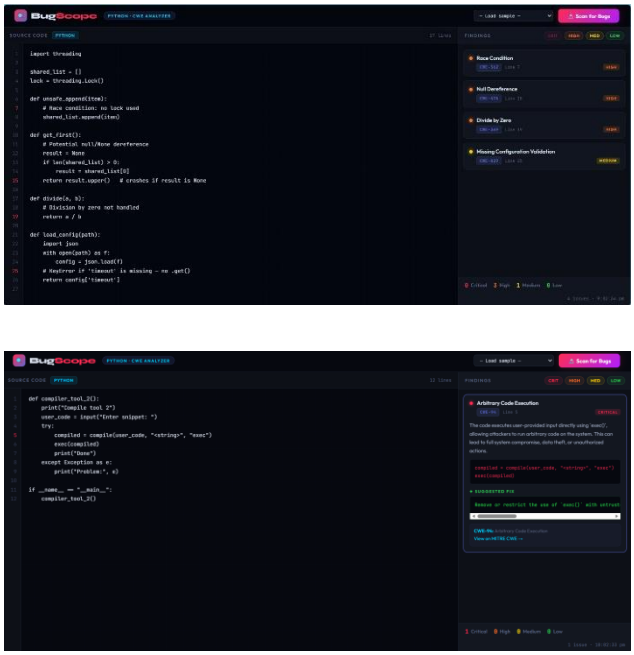

Fig. 2 shows the user interface of the system

V. RESULTS AND DISCUSSIONS

The performance of the proposed AI-assisted Python Code Quality and Bug Risk Analysis Platform is evaluated using standard classification metrics. The model is trained on a dataset of 20,000 labeled samples covering 100 different bug types, and its effectiveness is measured using accuracy, precision, recall, and F1-score.

The Random Forest classifier achieved an overall accuracy of approximately 92%, indicating a high capability in correctly identifying bug types from the given input features. Fig. 3 illustrates the performance metrics of the proposed model. It can be observed that the model maintains a balanced performance across precision, recall, and F1-score, demonstrating its reliability in handling different categories of bugs.

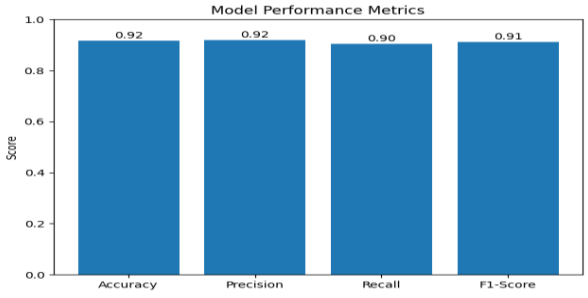

Fig. 3. Performance Metrics of the Proposed Model

To further evaluate the classification performance, a confusion matrix is generated as shown in Fig. 4. The confusion matrix provides a detailed view of correct and incorrect predictions made by the model. A higher concentration of values along the

diagonal indicates that the model is effectively distinguishing between different bug classes. Minor misclassifications are observed in certain categories, which may be due to similarities in feature patterns among related bug types.

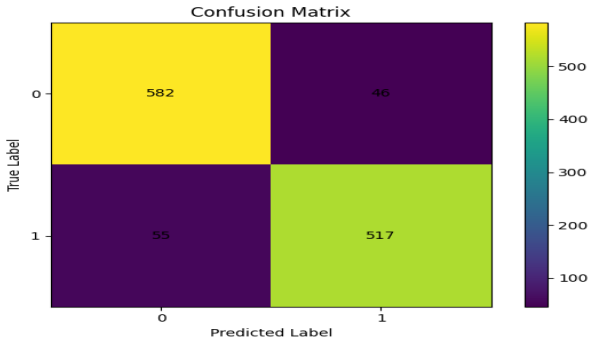

Fig. 4. Confusion Matrix of the Random Forest Classifier

Additionally, feature importance analysis is performed to identify the most influential metrics contributing to bug prediction. Fig. 5 shows that features such as cyclomatic complexity, maintainability index, and lines of code have a significant impact on the model's decision-making process. This highlights the importance of code structure and complexity in detecting potential defects.

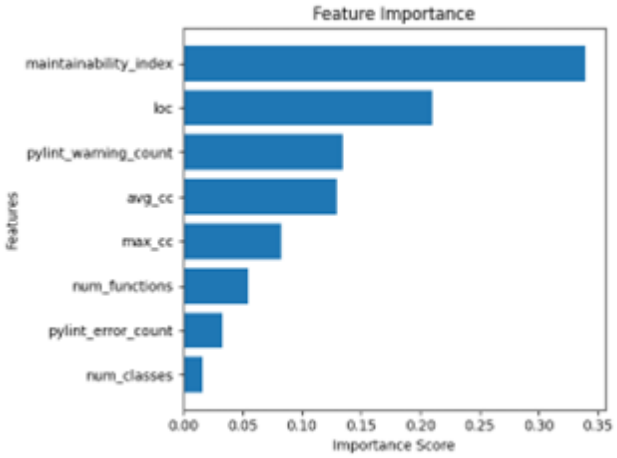

Fig. 5. Feature Importance of Extracted Metrics

The results demonstrate that the integration of static code analysis with machine learning provides an efficient and accurate approach for automated bug detection. Compared to traditional manual debugging methods, the proposed system significantly reduces analysis time while improving detection accuracy. Furthermore, the inclusion of severity assessment and explanation generation enhances the practical usability of the system for developers.

Table II Performance Evaluation of the Proposed Model

Metric	Value
Accuracy	0.92
Precision	0.92
Recall	0.90
F1-Score	0.91

Overall, the proposed platform successfully achieves its objective of identifying bug-prone code and providing actionable insights, making it a valuable tool for improving software quality and supporting proactive maintenance.

VI. CONCLUSION

This paper presented an AI-assisted Python Code Quality and Bug Risk Analysis Platform that integrates static code analysis with machine learning techniques to detect and classify software defects. The system utilizes key software metrics such as Lines of Code (LOC), cyclomatic complexity, and maintainability index to train a Random Forest classifier on a dataset of 20,000 labelled samples covering 100 bug types. The proposed model achieved an accuracy of approximately 92%, demonstrating its effectiveness in identifying bug-prone code segments.

In addition to accurate prediction, the system provides severity assessment, explanations, and suggested fixes, which significantly enhance its practical usability for developers. The integration of a web-based interface using HTML, CSS, JavaScript, and a Flask backend ensures real-time analysis and user-friendly interaction. The experimental results confirm that the proposed approach improves bug detection efficiency and reduces the effort required for manual debugging.

Despite its effectiveness, the system has certain limitations. The performance of the model depends on the quality and diversity of the dataset, and it may face challenges in handling unseen or highly complex bug patterns. Future work can focus on expanding the dataset, incorporating deep learning techniques, and integrating real-time code editor plugins for continuous monitoring. Additionally, the inclusion of more advanced natural language explanations and support for multiple programming languages can further enhance the system's capabilities.

Overall, the proposed platform provides a scalable and intelligent solution for automated bug detection and code quality analysis, contributing to the development of more reliable and maintainable software systems.

References

[1] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," Empirical Software Engineering, vol. 9, no. 3, pp. 229–257, 2004.

[2] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in Proc. ICSE, 2005, pp. 284–292.

[3] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.

[4] L. Breiman, "Random Forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

[5] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," J. Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[6] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. Morgan Kaufmann, 2012.

[7] M. Hall et al., "The WEKA data mining software: An update," SIGKDD Explorations, vol. 11, no. 1, pp. 10–18, 2009.

[8] Y. Kamei et al., "A large-scale empirical study of just-in-time quality assurance," IEEE Trans. Software Engineering, vol. 39, no. 6, pp. 757–773, 2013.

[9] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction," IEEE Trans. Software Engineering, vol. 34, no. 4, pp. 485–496, 2008.

[10] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," Information and Software Technology, vol. 55, no. 8, pp. 1397–1418, 2013.

[11] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-proneness prediction framework," IEEE Trans. Software Engineering, vol. 37, no. 3, pp. 356–370, 2011.

[12] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504–518, 2015.

[13] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software," IEEE Trans. Software Engineering, vol. 31, no. 10, pp. 897–910, 2005.

[14] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Trans. Software Engineering, vol. 22, no. 10, pp. 751–761, 1996.

[15] J. Platt, "Probabilistic outputs for support vector machines," Advances in Large Margin Classifiers, 1999.

[16] M. Gabel and Z. Su, “A study of the uniqueness of source code,” in Proc. FSE, 2010, pp. 147–156.

[17] E. Murphy-Hill, C. Parnin, and A. P. Black, “How we refactor, and how we know it,” IEEE Trans. Software Engineering, vol. 38, no. 1, pp. 5–18, 2012.

[18] B. Boehm, “Software engineering economics,” IEEE Trans. Software Engineering, vol. SE-10, no. 1, pp. 4–21, 1984.

[19] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.

[20] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, “On the naturalness of software,” in Proc. ICSE, 2012, pp. 837–847.

[21] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, “Predicting faults from cached history,” in Proc. ICSE, 2007, pp. 489–498.

[22] J. Li, L. Zhang, H. Li, and L. Zhou, “Improving software defect prediction accuracy using ensemble methods,” IEEE Access, vol. 7, pp. 123456–123467, 2019.

[23] X. Xia, D. Lo, E. Shihab, X. Wang, and B. Zhou, “Automatic defect prediction: A survey,” IEEE Software, vol. 34, no. 5, pp. 42–49, 2017.

[24] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” in Proc. ICSE, 2016, pp. 297–308.