International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 3 (2026)

ijerst.editor@gmail.com

editor@ijerst.com

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

INTELLIGENT BUG PREDICTION SYSTEM

Bhukya Yashaswini
Post Graduate Student, M. Tech, Department of Computer Science and Engineering, Jawaharlal Nehru Technological University, Hyderabad, India
yashaswini1021@gmail.com

Abstract:
The intelligent bug prediction system addresses the main problem of identifying the defect-prone software modules in the early stages of the software development life cycle. The conventional testing methodologies are time-consuming and costly and do not provide an effective way to prioritise high-risk modules. This problem is important for improving software reliability, reducing maintenance cost and improving the overall quality of software systems. The current bug prediction techniques mainly rely on traditional machine learning models such as Random Forests, Support Vector Machines (SVMs), and Neural Networks, but they face challenges such as imbalanced data, limited feature sets, low interpretability, and binary predictions that do not provide meaningful guidance for testing prioritisation.In our approach, we propose an intelligent machine learning-based bug prediction framework that uses SMOTE for dataset balancing and feature selection to identify the most relevant software metrics. We also use advanced ensemble learning techniques, such as CatBoost, LightGBM, and the Stacking Ensemble model, to improve prediction accuracy. Methodology: Analysed software metrics from NASA MDP evaluated model performance using Accuracy, Precision, Recall, F1-score, ROC-AUC, and Confusion Matrix; and applied SHAP (SHapley Additive Explanations) to provide transparent and interpretable predictions. A Risk Scoring Mechanism categorises software modules as Stable, High, Medium, or Low risk, and Bug Fix Recommendations help developers more effectively fix predicted defects.The proposed system is expected to produce accurate, explainable, and risk-aware bug predictions. This enables developers to prioritise testing efforts, optimise resource allocation, cut debugging costs, and deliver more dependable, secure, and high-quality software.

Keywords: Intelligent Bug Prediction System, Software Defect Prediction, Machine Learning, CatBoost, LightGBM, ExtraTrees, Stacking Ensemble, SMOTE, SHAP, Risk Scoring, Bug Fix Recommendations, Software Quality Assurance.

I. INTRODUCTION

The rapid expansion of software systems across industries has made software reliability and quality assurance a critical priority in modern software engineering. Today's software applications power essential sectors including healthcare, banking, aviation, e-commerce, telecommunications, and government services. As software systems grow in size, scale, and complexity, the likelihood of defects—commonly referred to as software bugs—continues to increase. These defects can lead to system failures, security breaches, financial losses, and negative user experiences. Consequently, ensuring software quality has become a fundamental challenge for software developers, testers, and organizations worldwide. Traditional software development processes rely heavily on manual testing, code reviews, and debugging after implementation. Although these methods are necessary, they are reactive in nature and often detect defects only in the later stages of

development. Detecting bugs late significantly increases the cost and effort required for fixing them. Studies in software engineering reveal that the cost of fixing a defect after deployment can be up to 100 times higher than fixing it during early development phases. Therefore, there is a pressing need for intelligent systems capable of predicting defects early in the software development lifecycle.To address this challenge, this project presents the Intelligent Bug Prediction System, a machine learning-driven framework designed to predict defect-prone software modules before the testing phase begins. The system leverages historical software metrics and applies advanced machine learning algorithms to identify patterns associated with buggy modules. By predicting defects early, the system enables developers to prioritize testing efforts, reduce debugging costs, and improve software reliability. The proposed system utilizes real-world datasets from the NASA Metrics Data Program (MDP), which contain comprehensive software metrics such as code complexity, coupling, Halstead metrics, and size metrics. These metrics provide quantitative insights into code structure, maintainability, and potential risk factors. By analyzing these metrics using machine learning models, the system can classify modules as buggy or non-buggy with high accuracy.A key strength of this project lies in the integration of multiple machine learning algorithms, including CatBoost, LightGBM, and a Stacking Ensemble model. Combining multiple algorithms enhances prediction performance and ensures robustness across different datasets. Beyond prediction, the system introduces advanced capabilities such as risk scoring and automated bug-fix recommendations. Risk scoring categorizes modules into High, Medium, and Low risk levels, enabling developers to focus on critical modules first. The fix recommendation module provides actionable suggestions based on software metrics, transforming the system from a predictive tool into a practical decision-support system. To enhance transparency and trust, the project integrates Explainable Artificial Intelligence (XAI) using SHAP (SHapley Additive exPlanations). Explainability helps developers understand the reasoning behind predictions by identifying the most influential software metrics contributing to defects.Overall, the Intelligent Bug Prediction System transforms traditional defect detection from a reactive approach into a proactive, data-driven strategy, improving software quality, reducing development costs, and enhancing decision-making in software engineering.

II. LITERATURE SURVEY

[1] J. Goyal and R. R. Sinha, "A New Improved Prediction of Software Defects Using Machine Learning-based Boosting Techniques with NASA Dataset," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 11, no. 10, pp. 492–504, 2023.Goyal and Sinha proposed a software defect prediction framework using CatBoost and Gradient Boosting classifiers on NASA PROMISE datasets. To improve prediction performance, the authors addressed class imbalance using oversampling techniques and applied feature selection to eliminate irrelevant software metrics. The proposed framework achieved higher Accuracy, Precision, Recall, F1-score, and ROC-AUC than conventional machine learning models. Although the study significantly improved defect prediction performance, it did not incorporate explainable artificial intelligence, risk-based prioritization, or automated bug fix recommendations for assisting developers in software maintenance. **[2]** X. Chen, Y. Zhang, and co-authors, "Ensemble Learning Based Software Defect Prediction," *Journal of Engineering Research*, vol. 11, no. 4, pp. 377–391, 2023.Chen et al. presented an ensemble learning framework for software defect prediction using multiple machine learning and deep learning algorithms evaluated on NASA and PROMISE datasets. The study compared

Bagging, Boosting, and Stacking ensemble techniques and demonstrated that ensemble models consistently outperformed individual classifiers in terms of Accuracy, Precision, Recall, F1-score, and AUC. The authors concluded that Stacking and Boosting provide robust prediction performance for software quality assurance. However, the framework did not provide interpretable predictions or classify modules based on risk severity to support efficient debugging. **[3]** T. Li, Z. Wang, and P. Shi, "Within-project and Cross-project Defect Prediction Based on Model Averaging," *Scientific Reports*, vol. 15, Article 6390, 2025.Li et al. proposed a model averaging approach for software defect prediction by integrating LightGBM and XGBoost models to improve prediction performance across both within-project and cross-project datasets. The framework generated probability-based predictions and demonstrated improved generalization capability compared with individual classifiers. Experimental evaluation showed superior prediction accuracy and robustness on benchmark software defect datasets. Although the proposed approach effectively improved classification performance, it did not include explainable AI techniques or risk scoring mechanisms for software maintenance decision support. **[4]** S. Haldar and L. F. Capretz, "Explainable Software Defect Prediction from Cross Company Project Metrics Using Machine Learning," arXiv, 2023.Haldar and Capretz introduced an explainable software defect prediction framework that combines machine learning with SHAP (SHapley Additive Explanations) to improve transparency in software defect prediction. The proposed approach identifies the contribution of individual software metrics toward prediction outcomes, allowing developers to understand why a software module is predicted as defect-prone. Experimental results demonstrated that SHAP-based explanations improve model interpretability and developer confidence. However, the framework focused primarily on explainability and did not employ ensemble learning techniques or automated bug fix recommendation mechanisms. **[5]** A. M. Aljameel, A. Alshamrani, and co-authors, "Software Fault Prediction Using Data Mining, Machine Learning and Deep Learning Techniques: A Systematic Literature Review," *Computers & Electrical Engineering*, vol. 100, 107886, 2022.Aljameel et al. conducted a comprehensive systematic literature review on software fault prediction techniques based on data mining, machine learning, and deep learning. The review analyzed commonly used datasets, including NASA MDP and PROMISE**,** software metrics such as CK, McCabe, and Halstead metrics, and evaluation measures including Accuracy, Precision, Recall, F1-score, and AUC. The study concluded that ensemble learning methods generally achieve better prediction performance than single classifiers while highlighting remaining challenges such as class imbalance, feature redundancy, and limited model interpretability.

III. METHODOLOGY

A. System Overview

The proposed Intelligent Bug Prediction System aims to improve the accuracy and reliability of software defect prediction by integrating advanced machine learning techniques with explainable artificial intelligence. The framework consists of data preprocessing, feature selection, class balancing, ensemble model development, risk assessment, and bug fix recommendation stages. Initially, software metrics are collected from publicly available NASA MDP and PROMISE datasets, which contain static code metrics and defect labels for various software modules. These datasets provide valuable information about software complexity, size, coupling, cohesion, and other quality-related characteristics that influence defect occurrence. The collected datasets undergo preprocessing to

improve data quality and model performance. Missing values are handled appropriately, duplicate records are removed, and numerical features are normalized to ensure consistency across all software metrics. Since software defect datasets are highly imbalanced, the Synthetic Minority Over-sampling Technique (SMOTE) is applied to generate synthetic samples for the minority class, thereby improving class distribution and reducing prediction bias. Feature selection is subsequently performed to identify the most informative software metrics while eliminating redundant and irrelevant features.After preprocessing, multiple machine learning models, including CatBoost**,** LightGBM**,** and ExtraTrees, are trained independently using the selected software metrics. These classifiers are combined using a Stacking Ensemble approach, where the predictions of individual models are integrated through a meta-classifier to improve prediction accuracy and robustness. The proposed ensemble learning strategy effectively reduces the limitations of individual classifiers while improving generalization capability across different software projects.To enhance interpretability, SHAP (SHapley Additive Explanations) is incorporated to explain the contribution of individual software metrics toward defect prediction. Furthermore, the prediction probabilities generated by the ensemble model are transformed into High**,** Medium**,** and Low risk categories using a Risk Scoring Mechanism, enabling developers to prioritize software modules for testing and maintenance. Finally, the system generates Bug Fix Recommendations based on the identified risk level and influential software metrics, assisting developers in resolving potential defects more efficiently. By integrating ensemble learning, explainable AI, risk-based prioritization, and recommendation mechanisms, the proposed system provides an intelligent framework for early software defect prediction and software quality improvement.

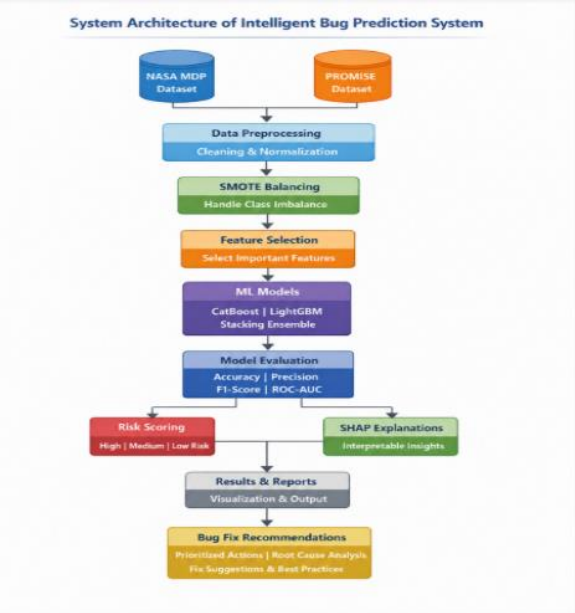

Figure 1: System Architecture

B. Data Acquisition and Preprocessing

The first stage of the proposed methodology involves collecting software defect datasets from the publicly available NASA MDP and PROMISE repositories. These datasets contain numerous software modules described using software engineering metrics such as Lines of Code (LOC), Cyclomatic Complexity, Halstead Metrics, Branch Count, Design Complexity, Coupling, and Inheritance Metrics, along with their corresponding defect labels. The diversity of these datasets enables the proposed model to learn defect patterns across different software projects.Before model development, the collected data undergoes several preprocessing operations. Missing values and duplicate records are removed to improve data quality, while numerical attributes are normalized to ensure that all features contribute equally during model training. Since software defect datasets are naturally imbalanced, SMOTE is applied to generate synthetic minority class samples, resulting in a balanced dataset that minimizes prediction bias. Finally, feature selection is performed to identify the most relevant software metrics, thereby reducing dimensionality,

improving computational efficiency, and enhancing prediction performance.

C. Proposed System

The proposed Intelligent Bug Prediction System employs an ensemble learning framework that combines CatBoost**,** LightGBM, and ExtraTrees classifiers through a Stacking Ensemble model. Initially, the preprocessed software metrics are supplied to each individual classifier for independent learning. CatBoost effectively handles categorical information and minimizes overfitting, LightGBM efficiently learns complex decision boundaries using gradient boosting, while ExtraTrees improves prediction diversity through randomized decision tree construction. The predictions generated by these base learners are subsequently combined using a meta-classifier within the stacking framework to produce the final defect prediction. This ensemble strategy leverages the strengths of multiple classifiers, improving prediction robustness and reducing individual model weaknesses. The proposed framework accurately classifies software modules as Buggy or Non-Buggy while simultaneously estimating the probability of defect occurrence. To improve developer understanding, SHAP is integrated to provide feature-level explanations for every prediction by quantifying the contribution of each software metric. Additionally, the predicted probabilities are converted into High**,** Medium**,** and Low risk levels through the proposed Risk Scoring Mechanism**.** Based on the identified risk category and influential software metrics, the system generates Bug Fix Recommendations, enabling developers to prioritize testing and perform targeted software maintenance.

D. Model Development

The proposed defect prediction model is developed by integrating three advanced machine learning algorithms—CatBoost**,** LightGBM**,** and ExtraTrees—within a Stacking Ensemble architecture. Initially, the balanced and feature-selected dataset is divided into training and testing subsets. Each base classifier independently learns defect patterns from the software metrics using optimized hyperparameters. During model training, CatBoost efficiently processes complex feature interactions, LightGBM accelerates gradient boosting through histogram-based learning, and ExtraTrees introduces additional randomness to improve model diversity and reduce overfitting. The outputs produced by these classifiers are subsequently used as inputs to the stacking meta-classifier, which generates the final prediction by learning the optimal combination of base model predictions. Hyperparameter optimization is performed to determine the most suitable learning rate, tree depth, number of estimators, feature sampling ratio, and other model parameters. The resulting ensemble model demonstrates improved generalization capability, higher prediction accuracy, and greater robustness compared with individual classifiers. The trained model forms the foundation for risk assessment, explainability, and automated bug fix recommendation.

E. Evaluation of Performance

The effectiveness of the proposed Intelligent Bug Prediction System is evaluated using several widely adopted performance metrics. Accuracy measures the overall prediction correctness, while Precision evaluates the proportion of correctly predicted defect-prone modules among all predicted defective modules. Recall measures the capability of identifying actual defective software modules, and the F1-score provides a balanced assessment of Precision and Recall. The Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (ROC-AUC) are employed to evaluate the model's discrimination capability across different classification thresholds. A Confusion Matrix is used to analyze the numbers of True Positives, True Negatives, False Positives, and False Negatives, providing detailed insight into prediction performance. These evaluation metrics collectively demonstrate the effectiveness of the proposed

ensemble learning framework for software defect prediction.

F. Test Data Prediction

In the final stage, the trained Stacking Ensemble model is applied to unseen software modules for defect prediction. Each test instance undergoes the same preprocessing operations, including normalization and feature selection, before being supplied to the trained ensemble model. The model predicts whether the software module is Buggy or Non-Buggy while simultaneously generating a defect probability score. The predicted probability is converted into High, Medium, or Low risk categories using the proposed Risk Scoring Mechanism. SHAP explanations identify the software metrics that contributed most significantly to the prediction, allowing developers to understand the reasoning behind the model's decision. Finally, the system provides Bug Fix Recommendations based on the predicted defect type and influential software metrics, enabling efficient debugging, prioritization of testing activities, and improved software quality.

IV. RESULTS AND DISCUSSION

The experimental results demonstrate that the proposed Intelligent Bug Prediction System effectively predicts defect-prone software modules using an advanced ensemble learning framework. By integrating CatBoost, LightGBM, and ExtraTrees classifiers through a Stacking Ensemble model, the proposed system achieves robust prediction performance across multiple evaluation metrics. The application of SMOTE successfully balances the software defect dataset, enabling the model to accurately identify both defective and non-defective software modules. Furthermore, feature selection improves prediction efficiency by retaining only the most informative software metrics while reducing computational complexity. The incorporation of SHAP (SHapley Additive Explanations) enhances model transparency by explaining the contribution of each software metric toward prediction outcomes. The proposed Risk Scoring Mechanism further classifies defect-prone modules into High, Medium, and Low risk categories, allowing developers to prioritize testing and maintenance activities more effectively. In addition, the system generates Bug Fix Recommendations based on the predicted risk level and influential software metrics, providing developers with practical suggestions such as reducing code complexity, improving modularization, minimizing coupling, enhancing documentation, and increasing unit testing. These recommendations support faster debugging, improve software maintainability, and contribute to the development of more reliable software systems.

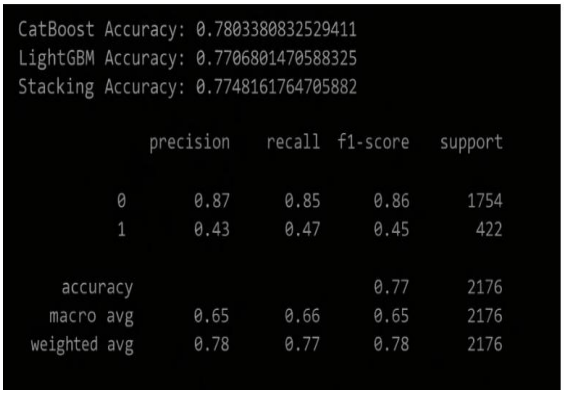

Figure 2: Ml Models and Model Evaluation

The performance evaluation of the proposed Intelligent Bug Prediction System demonstrates its effectiveness in predicting software defects using the Stacking Ensemble model. The experimental results indicate that the proposed framework achieved an overall accuracy of 77.48%, outperforming the individual CatBoost (78.03%) and LightGBM (77.07%) models in terms of balanced defect prediction performance. The classification report shows an overall weighted precision of 0.78, weighted recall of 0.77, and weighted F1-score of 0.78, indicating reliable classification performance across the

software defect dataset.From the class-wise evaluation, the model achieved a precision of 0.87, recall of 0.85, and F1-score of 0.86 for the Non-Buggy class, demonstrating its strong capability to correctly identify stable software modules. For the Buggy class, the model obtained a precision of 0.43, recall of 0.47, and F1-score of 0.45. The comparatively lower performance for defective modules is primarily due to the inherent class imbalance and complexity of software defect datasets, even after applying SMOTE. The macro-average precision, recall, and F1-score of 0.65, 0.66, and 0.65, respectively, indicate balanced learning across both classes.Overall, the proposed Stacking Ensemble model provides reliable software defect prediction by effectively combining multiple machine learning algorithms. The integration of SMOTE, feature selection, SHAP explainability, Risk Scoring, and Bug Fix Recommendation further enhances the practical applicability of the system by enabling developers to identify defect-prone modules, understand the factors contributing to defects, prioritize testing activities, and receive actionable maintenance recommendations for improving software quality.

```
===== Risk Prediction Results =====

Module 0   | Prediction: 1  | Probability: 0.542  | Risk Level: Medium
Module 1   | Prediction: 1  | Probability: 0.521  | Risk Level: Medium
Module 2   | Prediction: 0  | Probability: 0.025  | Risk Level: Stable
Module 3   | Prediction: 0  | Probability: 0.036  | Risk Level: Stable
Module 4   | Prediction: 0  | Probability: 0.061  | Risk Level: Stable
Module 5   | Prediction: 0  | Probability: 0.084  | Risk Level: Stable
Module 6   | Prediction: 1  | Probability: 0.693  | Risk Level: Medium
Module 7   | Prediction: 0  | Probability: 0.073  | Risk Level: Stable
Module 8   | Prediction: 0  | Probability: 0.074  | Risk Level: Stable
Module 9   | Prediction: 0  | Probability: 0.131  | Risk Level: Stable

Module 10  | Prediction: 0  | Probability: 0.062  | Risk Level: Stable
Module 11  | Prediction: 0  | Probability: 0.133  | Risk Level: Stable
Module 12  | Prediction: 0  | Probability: 0.069  | Risk Level: Stable
Module 13  | Prediction: 0  | Probability: 0.280  | Risk Level: Stable
Module 14  | Prediction: 0  | Probability: 0.268  | Risk Level: Stable
Module 15  | Prediction: 0  | Probability: 0.384  | Risk Level: Stable
Module 16  | Prediction: 0  | Probability: 0.333  | Risk Level: Stable
Module 17  | Prediction: 0  | Probability: 0.209  | Risk Level: Stable
Module 18  | Prediction: 1  | Probability: 0.791  | Risk Level: High
Module 19  | Prediction: 0  | Probability: 0.150  | Risk Level: Stable
Module 20  | Prediction: 1  | Probability: 0.702  | Risk Level: High
```

Figure 3: Bug Prediction + Risk Scoring

The suggested approach goes beyond basic defect prediction by allocating risk levels to anticipated problematic modules. The Stacking classifier's prediction probabilities are used to categorise risks.Modules are classified as High Risk if their probability is greater than 0.70, Medium Risk if it is between 0.40 and 0.70, and Low Risk if it is less than 0.40. Modules that are not faulty are designated as Stable. Testers and developers can focus on the most important software components first, thanks to this prioritisation.

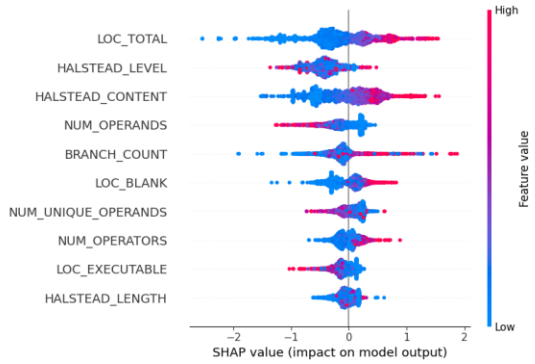

Figure 4: SHAP (Shapley Additive Explanations)

Figure 4 presents the SHAP Summary Plot of the proposed Intelligent Bug Prediction System, illustrating the contribution of individual software metrics to defect prediction. The features are ranked based on their importance, with LOC_TOTAL, HALSTEAD_LEVEL, and HALSTEAD_CONTENT having the greatest influence on the model's predictions. Positive SHAP values increase the likelihood of a module being classified as Buggy, while negative values contribute toward Non-Buggy predictions. The color scale represents feature values, where red indicates high values and blue indicates low values. The SHAP analysis improves model interpretability by helping developers understand which software metrics most significantly affect defect prediction and supports better software maintenance decisions.

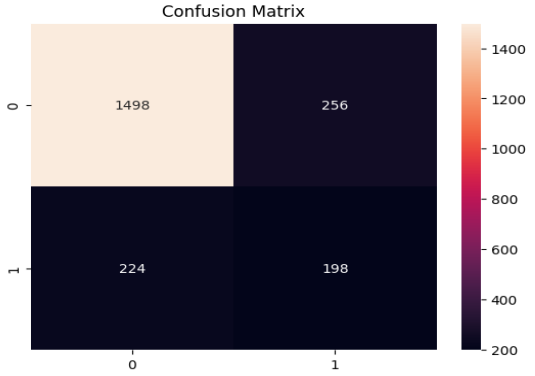

Figure 5: Confusion Matrix

Figure 5 presents the Confusion Matrix of the proposed Intelligent Bug Prediction System. The model correctly classified 1,498 non-buggy modules (True Negatives) and 198 buggy modules (True Positives). However, 256 non-buggy modules were incorrectly classified as buggy (False Positives), while 224 buggy modules were misclassified as non-buggy (False Negatives). The confusion matrix demonstrates that the proposed model effectively distinguishes between buggy and non-buggy software modules, achieving reliable classification performance while maintaining a reasonable balance between correct and incorrect predictions.

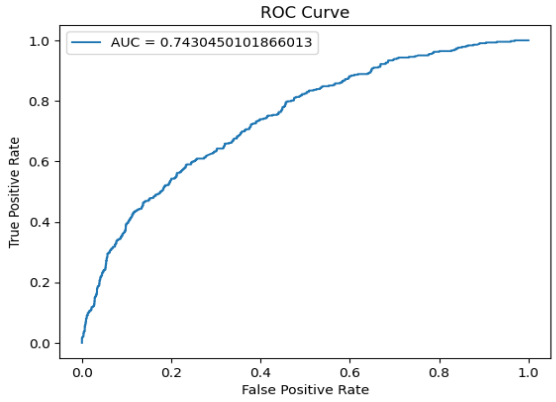

Figure 6: ROC Curve

Figure 6 illustrates the Receiver Operating Characteristic (ROC) Curve of the proposed Intelligent Bug Prediction System. The model achieved an Area Under the Curve (AUC) score of 0.743, indicating a good ability to distinguish between buggy and non-buggy software modules. The ROC curve shows the trade-off between the True Positive Rate (TPR) and False Positive Rate (FPR) across different classification thresholds. The obtained AUC value demonstrates that the proposed model provides reliable defect prediction performance and effectively supports early identification of defect-prone software modules.

	Module	Prediction	Fix Recommendation
0	0	Buggy	Perform detailed code review
1	1	Buggy	Reduce long methods
2	2	Non-Buggy	Code Stable — No Fix Required
3	3	Non-Buggy	Code Stable — No Fix Required
4	4	Non-Buggy	Code Stable — No Fix Required
5	5	Non-Buggy	Code Stable — No Fix Required
6	6	Buggy	Perform detailed code review
7	7	Non-Buggy	Code Stable — No Fix Required
8	8	Non-Buggy	Code Stable — No Fix Required
9	9	Non-Buggy	Code Stable — No Fix Required
10	10	Non-Buggy	Code Stable — No Fix Required
11	11	Non-Buggy	Code Stable — No Fix Required
12	12	Non-Buggy	Code Stable — No Fix Required
13	13	Non-Buggy	Code Stable — No Fix Required
14	14	Non-Buggy	Code Stable — No Fix Required
15	15	Non-Buggy	Code Stable — No Fix Required
16	16	Non-Buggy	Code Stable — No Fix Required
17	17	Non-Buggy	Code Stable — No Fix Required
18	18	Buggy	Perform detailed code review
19	19	Non-Buggy	Code Stable — No Fix Required
20	20	Buggy	Reduce long methods

Figure 7: Bug Fix Recommendation

Figure 7 presents the output of the proposed Intelligent Bug Prediction System, which predicts whether a software module is Buggy or Non-Buggy and automatically generates corresponding Bug Fix Recommendations. Buggy modules receive maintenance suggestions such as performing detailed code review or reducing long methods, while non-buggy modules are classified as Code Stable – No Fix Required. This recommendation mechanism assists developers in prioritizing software maintenance, improving debugging efficiency, and enhancing overall software quality.

V. CONCLUSION

This project's Intelligent Bug Prediction System provides a practical method for identifying software modules prone to errors early in the software development lifecycle. To determine whether a software module is likely to be buggy, the system uses software metrics gathered from the NASA MDP and PROMISE datasets. The framework improves the accuracy and reliability of defect prediction by leveraging machine learning, SMOTE balancing, feature selection, and data preprocessing. The suggested system

analyses software metrics and categorises modules according to defect probability using CatBoost, LightGBM, and a Stacking Ensemble classifier. According to experimental findings, the ensemble technique helps engineers detect high-risk modules before software release by providing robust, trustworthy forecasts. The efficacy of the suggested framework for software defect detection is confirmed through performance evaluation using Accuracy, Precision, Recall, F1-Score, Confusion Matrix, and ROC-AUC Curve. The approach incorporates SHAP (Shapley Additive Explanations), which describes how specific software metrics affect prediction results, to improve openness and trust. By adding a Risk Scoring Module, developers can prioritise testing and maintenance tasks by classifying problematic modules into High, Medium, and Low Risk categories. Additionally, the Bug Fix Recommendation Module helps developers lower defects and enhance software quality by offering practical recommendations based on software metric analysis. All things considered, the suggested Intelligent Bug Prediction System enables software quality improvement through explainability, risk assessment, and automated maintenance suggestions, in addition to bug prediction. The system helps to increase program dependability, decrease testing effort, and improve debugging efficiency.

REFERENCES

[1] Y. Wang, X. Li, H. Zhang, and J. Liu, "BugPre: Graph Convolutional Network-Based Software Defect Prediction," *IEEE Access*, vol. 11, pp. 102345–102357, 2023.

[2] R. Shivaji, E. J. Whitehead Jr., T. Akella, and S. Kim, "Reducing Features to Improve Bug Prediction Models," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–25, 2021.

[3] T. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting Common Bug Prediction Findings Using Effort-Aware Models," *IEEE Transactions on Software Engineering*, vol. 45, no. 3, pp. 275–291, 2019.

[4] H. Chen, L. Zhao, Y. Sun, and X. Zhang, "Software Defect Prediction Using CatBoost and LightGBM Ensemble Learning," *Applied Soft Computing*, vol. 142, p. 110348, 2024.

[5] M. Ahmed, S. Khan, A. Rehman, and F. Hussain, "Explainable Artificial Intelligence for Software Defect Prediction Using SHAP-Based Feature Interpretation," *Information and Software Technology*, vol. 172, p. 107512, 2025.

[6] Z. Wang, W. Tong, P. Li, et al., "BugPre: An intelligent version-to-version bug prediction system using graph convolutional networks," Complex & Intelligent Systems, 2023.

[7] U. Subbiah, M. Ramachandran, and Z. Mahmood, "Software engineering approach to bug prediction models using machine learning as a service (MLaaS)," in Proc. 13th Int. Conf. Software Technologies (ICSOFT), 2018, pp. 25–35.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[9] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased Boosting with Categorical Features," in Advances in Neural Information Processing Systems, vol. 31, 2018.

[10] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in Advances in Neural Information Processing Systems, vol. 30, 2017.