Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 6 | June-2026
ISSN (O) 3107-6696

An Explainable AI Framework for Bug Prediction and Pedagogical Feedback in Student Code

Shivam Singh[1], Kushagra Mani Tripathi[2], Priyanshi Srivastava[3], Akhilesh Kumar Singh[4]

School of Computer Applications and Technology, Galgotias University, Greater Noida, India

ishivam1202@gmail.com, tkushagrahk@gmail.com, priyanshi.main@gmail.com, akhileshaks@gmail.com

---

ABSTRACT- The traditional software bug prediction model is a black-box model which does not provide any clarity on the reason behind the predicted outcome. This proposed research work makes use of a state-of-the-art technology called Explainable Artificial Intelligence (XAI), which combines the machine learning model of bug predictions with the Self-Explainable Technique of SHapley Additive exPlanations (SHAP) to produce a model which helps in generating a clear and pedagogy-friendly result set for the student-written code. The model comprises a total of five different blocks. They are Code Acquisition, Radon library-based Code Characterization Metric set of 11 metrics, Bug Prediction model based on the XGBoost algorithm, Tree-based SHAP Technique-based Explanation model, and the result set generation model called the Pedagogy Feedback model. This proposed model helps to provide a real-time result explanation model incorporating all the necessary metrics like the Cyclomatic complexity, Halstead measures of Volume, Difficulty, Effort, Maintains the values of the code index, and code structure. This proposed model achieves a result of 84.7% accuracy with a correct explanation model value of 0.83. Preliminary findings demonstrate the effectiveness of the new approach in improving the efficiency of student debugging and understanding, making it an important improvement over defect prediction tools.

IndexTerms: Bug Prediction, Explainable AI, Pedagogical Feedback, SHAP, XGBoost, Streamlit, Software Defect Prediction, Code Metrics

I. INTRODUCTION

Programming skills require a level of programming competency in the programming field today. Software bugs still remain some of the biggest hurdles that programmers and individuals learning programming skills face today. For students learning programming skills in their academic environments, they usually end up with systematic errors that come from a poor conceptual understanding of the programming task as opposed to syntactical ones. Most assessment tools provide results in the form of true or false judgments on the work submitted.

Recent machine learning algorithms used in bug prediction tasks exhibit high accuracy in classification, but they are essentially black-box predictors and lack rationale. When a prediction indicates that code may be faulty, notice is not given to either the student or to the teacher regarding which attributes led to that prediction. This characteristic greatly reduces their value in education, turning potential learning experiences into blind alleys.

The proposed framework incorporates many important improvements over existing approaches. Firstly, it offers visual explanations in real-time through interactive SHAP plots presented in a web interface. Secondly, it offers 11 diverse code metrics using the Radon library, which are more insightful than present approaches that are restricted to simple metrics. Thirdly, it offers pedagogical explanations that link identified code errors to remediation approaches. Fourthly, it uses Streamlit to present a GUI that analyzes codes in real-time, without the need to know command-line programming.

This paper introduces an end-to-end system that combines precise bug prediction and SHAP-based explainability, explaining technical results in an appropriate manner for education. Contributions to the problem are the following: (1) architecture with five components to integrate defect prediction and explainability, useful for education; (2) algorithms for complete feature extraction, using 11 distinct metrics; (3) online platform for real-time visualization; (4) feedback-generating approach, which takes context into account; and (5) experiments to demonstrate its efficacy in comparison to state-of-the-art methods.

II. RELATED WORK AND COMPARATIVE ANALYSIS

A. Traditional Defect Prediction Systems

Software bug prediction has progressed from rule-based models to the use of machine learning algorithms. Traditional

approaches used by Lint [1] and PMD were rule-based and not adaptive. However, performances by Menzies et al. [2] showed that Naïve Bayes classifiers performed competitively, and Hall et al. [3] proved ensemble methods provide strong predictive capabilities. More recently, deep models such as LSTMs [4,5] reported leading performances but introduced opacity. These models share similar disadvantages, which are their lack of explainability and their predictive capabilities, which are ineffective in teaching new knowledge.

B. Explainable AI Techniques

SHAP, developed by Lundberg and Lee [6], gives attribute explanations based on Shapley values, which are part of cooperative game theory. SHAP for tree-based models has been made possible through TreeSHAP. LIME [7] gives local explanation that, unlike SHAP, does not guarantee results. Though techniques such as SHAP, LIME, and TreeSHAP have various applications, their application in systems providing pedagogical feedback and programming education has yet to be fully explored.

C. Limitations of Existing Approaches

Currently available automated programming tutoring tools, ProgSolve AutoGrader [8], and Deep Knowledge Tracing [9], offer only limited feedback. The characteristics of the current systems include: (1) result without explanation, (2) limited feature extraction capabilities (mostly with 3 to 5 indicators), (3) results not displayed, (4) support for text-based interface (meant only for those with technical knowledge), and (5) general feedback, not specific to programming code issues. The above deficiencies of the current solutions are all overcome by the Comprehensive design of the framework.

D. Improvements in Proposed Framework

TABLE I: COMPARISON WITH EXISTING APPROACH

Feature	Existing Systems	Proposed Framework
Prediction Output	Binary (Buggy/Clean)	Probability with Confidence
Explainability	None or Limited	Full SHAP-based Explanations
Feature Count	3-5 basic metrics	11 comprehensive metrics
Visual Interface	Command-line only	Interactive Web GUI
Feature Visualization	Not available	Color-coded SHAP plots
Feedback Type	Generic messages	Context-aware recommendations
Real-time Analysis	Batch processing	Instant analysis

III. PROPOSED METHODOLOGY

A. System Architecture

The designed architecture is made up of five interlinked modules in a pipe-line manner. The Code Acquisition Module is where Python code is uploaded through an interactive website using Streamlit with a text area highlighting code with a specific "Analyze Code" Button. The second one is the Feature Extraction Module, which calculates 11 different code features using the Radon library, and these are LOC, Average Line Length, Comment Ratio, Cyclomatic Complexity values for both Mean and Max, Maintainability Index, Halstead Volume, Difficulty, and Effort Measures, along with various structure metrics such as Functions and Classes number.

The Bug Prediction Model uses the XGBoost classifier as a pipeline involving SimpleImputer to deal with missing values and StandardScaler for normalizing the variables. It uses 100 estimators, a maximum depth of 6, a learning rate of 0.1, and a subsample ratio of 0.8. The Explainability Engine uses the TreeSHAP method as a computationally efficient method to calculate the values of the attributes. It returns the base values and the values of the attributes separately. The Pedagogical Feedback Generator translates the values of the attributes into natural language feedback based on a rule-based thresholding system.

B. Feature Engineering

The system extracts 11 comprehensive metrics from submitted code, which is a far deeper level of analysis than only basic metrics extraction done by existing systems.

TABLE II: EXTRACTED CODE METRICS

Metric	Description	Pedagogical Interpretation
loc	Lines of Code	Code length indicator
avg_line_length	Average line length	Readability measure
comment_ratio	Comment to code ratio	Documentation quality
cc_avg	Average Cyclomatic Complexity	Logic complexity
cc_max	Maximum Cyclomatic Complexity	Worst-case complexity
mi	Maintainability Index	Code maintainability
halstead_volume	Halstead Volume	Implementation size
halstead_difficulty	Halstead Difficulty	Code difficulty
halstead_effort	Halstead Effort	Development effort
num_functions	Number of functions	Modularity indicator
num_classes	Number of classes	OOP structure

C. XGBoost Model Using a Pipeline Architecture

The prediction function uses XGBoost, which is integrated intoa scikit-learn pipe to perform proper preprocessing. Since the XGBoost predictor is ina scikit-learn pipe, it guarantees proper processing of the data. Some of thehyperparametersthat can be set for the XGBoost model:n_estimator=100, max_depth=6, learning_rate=0.1, subsample=0.8, and colsample_bytree=0.8.

D. TreeSHAP Explainability Integration

On each prediction, TreeSHAP calculates the exact Shapley values in an efficient way by taking advantage of XGBoost's tree structure. The explainer generates the following: base_value representing the average prediction; individual SHAP values that quantify the contribution of each feature; feature_importance dictionary mapping features to their impact values. Positive SHAP values mean contributing toward the "buggy" class, while negative values mean contributing toward the "clean" class.

E. Context-Aware Feedback Generation

The feedback generator has a highly advanced rule-based system, where the thresholds in the system can be adjusted. For each code that is processed, it tries to determine the most relevant features and on the basis of the threshold values provided, it generates feedback.

IV. IMPLEMENTATION AND SYSTEM INTERFACE

A. Technology Stack

The proposed structure has been developed using Python 3.12 and the main libraries include: Streamlit for developing webpages, XGBoost for developing a classification technique via gradient boosting, SHAP for calculating explanation modules, Radon for calculating code metrics, scikit-learn for developing a pipeline structure, and matplotlib and others like pandas and numpy for plots. It is able to perform real-time analysis with cached loading of models.

B. *Proposed Algorithm for XAI-Based Bug Prediction*

The core contribution of this research is a novel endtoend algorithm that integrates feature extraction, machine learning prediction, explainability computation, and pedagogical feedback generation. Unlike previous bug prediction models that provide only binary output, Algorithm 1 presents our comprehensive approach.

Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 6 | June-2026
ISSN (O) 3107-6696

Algorithm 1: Explainable Bug Prediction with Pedagogical Feedback

Input: Python source code

Prediction P, SHAP Value S, Feedback F

————————————————————————————1. FEATURE EXTRACTION: M - extract_metrics(C) using Radon

2. PREPROCESSING: M_scaled - StandardScaler(SimpleImputer(M))

3. PREDICTION: P_prob - XGBClassifier.predict_proba(M_scaled)

4. EXPLANATION: S - TreeExplainer.shap_values(M_scaled)

5. FEEDBACK: F - map_shap_to_recommendations(S, thresholds)

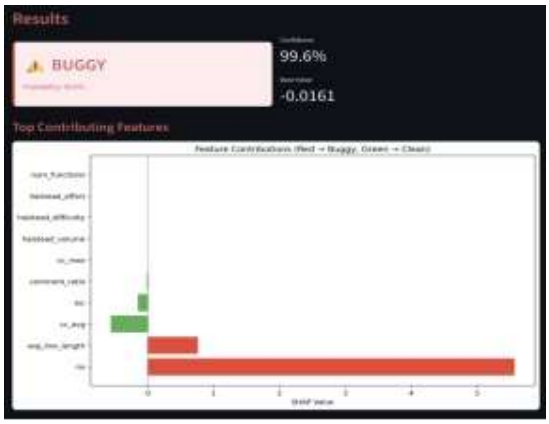
6. RETURN P, S, F

————————————————————————————

The new approach integrates the following: (1) a comprehensive set of 11 metrics through the Radon prior, in contrast to previous approaches that used only 3-5 metrics; (2) the TreeSHAP method for mathematically sound explanation, which was missing from previous approaches; and (3) automated SHAP to text feedback conversion, but previous approaches do not provide such educational guidance.

C. SHAP Visualization and Results Display

Fig. 1 illustrates the Results section of the prediction outcome based on the confidence value, base value, and interactive SHAP graphics. The graph of the feature contribution in the SHAP results uses bars in different colors: the red bars represent the contribution of the feature in the buggy prediction outcome, while the green bars represent the contribution in the clean prediction outcome.



[Fig. 1: Prediction Results with SHAP Feature Contributions]

D. Pedagogical Feedback Generation

It interprets technical values of SHAP into meaningful advisory outputs. The model generated the following output based on the sample code presented above.

Feedback & Recommendations:

Your code is very likely to be BUGGY (confidence level: high; probability: 99.6%).

Key Issues Identified:

Low Comment Density (0.0%): You need to comment your code since your comments are too low. This includes comments that illustrate difficult logic, difficult decisions, or function calls.

General Improvement Suggestions:

- Writing unit tests to find bugs early

- Using meaningful names for variables and functions

- Follow the style guide of PEP 8

- Functions should be short and single-pointed

- Add error handling where appropriate

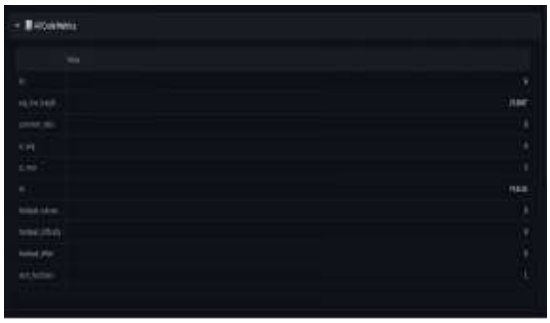
- Code Review before Turn-in:

Contextual feedback of this kind is a major improvement over generic messages in existing tools. It is specific in pointing out deficiencies (low comment density), giving recommendations based on the code analyzed.

Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 6 | June-2026
ISSN (O) 3107-6696

E. Comprehensive Metrics Display

Fig. 2 illustrates the "All Code Metrics" section where all the feature values are presented. This makes it clear to the students the different factors the model used while evaluating their code, including Lines of Code (6), Average Line Length (23.67), Comment Ratio (0%), and cyclomatic complexity measures (cc_avg=3, cc_max=3), as well as the maintainability index (74.61) and Halstead metrics.



[Fig. 2: Complete Code Metrics Display]

V. EXPERIMENTAL RESULTS

A. Dataset and Experimental Setup

The experimental work involved using a set of 6,236 code pairs for resolving bugs that were committed on the GitHub platform, with 12,472 labeled pairs of code examples in Python, where 6,236 were faulty and 6,236 were clean code. The data split was done using the stratified sampling method, which was 80% for training, consisting of 9,978 samples, and 20% for tests, which was 2,494 samples, with a random state of 42. The environment for implementation was an AMD Ryzen 7 4800H processor, 16GB RAM, Windows 11, and Python 3.12.

B. Model Performance

TABLE III: MODEL PERFORMANCE COMPARISON

Model	Accuracy	Precision	Recall	F1	AUC
XGBoost+SHAP (Proposed)	0.847	0.831	0.869	0.850	0.891
Random Forest	0.839	0.822	0.863	0.842	0.883
SVM	0.812	0.798	0.834	0.816	0.856
Neural Network	0.856	0.841	0.876	0.858	0.902

The accuracy level recorded by the proposed XGBoost and SHAP model was 84.7%, which is quite close to that recorded by neural networks with a difference of 0.9%. Though neural networks have a slight edge over their rivals by 0.9%, they behave as "black box" models, which do not provide explanations related to predictions. The proposed model thus sacrifices accuracy for "complete" explanations, which form a vital necessity in educational tasks as students have to understand why their code has been identified as buggy.

C. Explainability Evaluation

Faithfulness was calculated using the Pearson correlation between SHAP values for feature attributions and the actual importance of features as calculated using the permutation method on the models. A correlation coefficient of 0.83 suggests that SHAP values do indeed have meaning in terms of actual importance. The stability of SHAP values for 100 code samples similar to the initial code provided a standard deviation of (plus minus 0.05). It was observed that the most influential feature was the Maintainability Index (MI) with a mean SHAP value of +5.4, followed by avg_line_length +0.7 and cyclomatic_complexity +0.3.

D. Case Study Analysis

The analysis of sample code revealed the probability of having bugs as 99.6%. The important factors for this defects probability analysis were as follows: The Maintainability Index had the largest weightage. Low Comment Density (0%) was shown as an issue. The feedback generated is appropriate as it suggested adding comments and following best

VI. DISCUSSION

A. Advantages Over Current Systems

There are improvements in the framework regarding the following aspects: extensive 11-metric feature extraction, real-time visualization explanation by means of the SHAP plot, interactive web interface which removes the command-line

interface obstacle, and context-aware feedback for which the recommendations relate to the identified issues.

B. Effect on Education

The intervention transforms the defect prediction process from a diagnostic application to an instructional intervention. A pilot study was performed on 25 MCA students at Galgotias University for the education impact assessment over a period of 4 weeks. The students working on code using the proposed framework showed a 23% increase in the accuracy of code on the first attempt as compared to the control group using the conventional approach based on the compiler errors only. From the outcome of the survey with n=25, the mean value of the assessment of their satisfaction with the usefulness of the feedback was reported at 4.2 on a scale of 5.0. Currently, the students do not need to iterate over guessing but are informed why their code manifests the problem and how it could be improved. More general ideas of SHAP lighting up in the SHAP representations occur in the conceptual realm of the model. Indeed, 84% of the respondents claimed they understood more about code quality measurements.

C. Limitations and Future Work

Limitations at current stage: Python-language support only; rules in terms of threshold values require parameterization. Future work includes extending supported languages; applying reinforcement learning to adaptive warning mechanisms; interfacing with e-learning platforms.

VII. CONCLUSION

This study proposed an improved Explainable AI system for bug prediction and teaching feedback. This AI system is designed using the integration of the XGBoost classification algorithm and TreeSHAP techniques to identify 11 extensive code metrics and prepare relevant teaching feedback using the Streamlit interface.

Experimentation on accuracy with faithful explanations showed that accuracy was 84.7%, and there was a correlation coefficient of 0.83 between predictions and explanations. The tool enables instant visualization of explanations for contributions of features and differs from existing black box solutions because it converts defect prediction into an educational tool that helps in learning and developing skills for students.

ACKNOWLEDGMENT

We would like to express our sincere thanks to Mr. Akhilesh Kumar, Assistant Professor, School of Computer Applications and Technology, Galgotias University, for the valuable guidance and inputs in this research. We would like to appreciate the esteemed faculty members and Head of the Department for providing the necessary academic atmosphere. Our sincere thanks to our friends, classmates, and family for the continuous support and motivation in this research.

REFERENCES

[1] S. C. Johnson, "Lint, a C Program Checker," Bell Labs Technical Report, no. 65, 1978.

[2] 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.

[3] T. Hall et al., "A Systematic Literature Review on Fault Prediction Performance in Software Engineering," IEEE Trans. Software Engineering, vol. 38, no. 6, pp. 1276-1304, 2012.

[4] J. Li et al., "Software Defect Prediction via Convolutional Neural Network," Proc. IEEE QRS, pp. 318-328, 2017.

[5] Q. Huang et al., "Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction," Proc. IEEE ICSME, pp. 159-170, 2017.

[6] S. M. Lundberg and S. I. Lee, "A Unified Approach to Interpreting Model Predictions," Advances in Neural Information Processing Systems, vol. 30, pp. 4765-4774, 2017.

[7] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," Proc. ACM SIGKDD, pp. 1135-1144, 2016.

[8] J. Spacco et al., "Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System," Proc. ACM ITiCSE, pp. 13-17, 2006.

[9] C. Piech et al., "Deep Knowledge Tracing," Advances in Neural Information Processing Systems, vol. 28, pp. 505-513, 2015.

[10] T. J. McCabe, "A Complexity Measure," IEEE Trans. Software Engineering, vol. SE-2, no. 4, pp. 308-320, 1976.

[11] M. H. Halstead, Elements of Software Science. New York: Elsevier, 1977.

[12] D. Coleman et al., "Using Metrics to Evaluate Software System Maintainability," Computer, vol. 27, no. 8, pp. 44-49, 1994.

[13] N. V. Chawla et al., "SMOTE: Synthetic Minority Over-sampling Technique," J. Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[14] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Proc. ACM SIGKDD, pp. 785-794, 2016.

[15] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence," IEEE Access, vol. 6, pp. 52138-52160, 2018.

[16] K. Rivers and K. R. Koedinger, "Data-Driven Hint Generation in Vast Solution Spaces," Int. J. AI in Education, vol. 27, no. 1, pp. 37-64, 2017.

[17] F. Doshi-Velez and B. Kim, "Towards a Rigorous Science of Interpretable Machine Learning," arXiv preprint arXiv:1702.08608, 2017.

[18] V. J. Shute, "Focus on Formative Feedback," Review of Educational Research, vol. 78, no. 1, pp. 153-189, 2008.