# A Comparative Study of Explainable AI Techniques for Software Defect Prediction

*An empirical analysis for Software defect prediction using XAI


Balaji V
*Department of Computer Science and Engineering*
*Amrita School of Computing, Coimbatore*
*Amrita Vishwa Vidyapeetham, India*
cb.en.u4cse20211@cb.students.amrita.edu

Kennith Ronnie E
*Department of Computer Science and Engineering*
*Amrita School of Computing, Coimbatore*
*Amrita Vishwa Vidyapeetham, India*
cb.en.u4cse20230@cb.students.amrita.edu

Sailesh G
*Department of Computer Science and Engineering*
*Amrita School of Computing, Coimbatore*
*Amrita Vishwa Vidyapeetham, India*
cb.en.u4cse20253@cb.students.amrita.edu

Prasanna M
*Department of Computer Science and Engineering*
*Amrita School of Computing, Coimbatore*
*Amrita Vishwa Vidyapeetham, India*
cb.en.u4cse20247@cb.students.amrita.edu

Sabarish B A
*Department of Computer Science and Engineering*
*Amrita School of Computing, Coimbatore*
*Amrita Vishwa Vidyapeetham, India*
ba_sabarish@cb.amrita.edu



***Abstract*—This study aims to make software predictive models using special methods called explainable AI for better understanding for researchers. The research emphasizes the importance of predictive models in identifying potential bugs in software systems due to their intricate nature and challenges post-deployment. The complexity of certain machine learning models necessitates understanding of their decision-making mechanisms among stakeholders.. Leveraging techniques such as Recursive Feature Elimination (RFE), Local Interpretable Model-agnostic Explanations (LIME), SHapley Additive exPlanations (SHAP), and PyExplainer, explores the impact of feature value fluctuations on software defect likelihood, addressing key research questions on feature significance, individual prediction explanations, and global model behavior. The study analyzes the Apache Ant dataset to demonstrate the effectiveness of various methodologies in identifying critical features, explaining individual/global predictions, and facilitating interactive model exploration. The research emphasizes the significant role of XAI methodologies in enhancing model, enabling users to easily identify bugs and deploy robust software systems.**



***Index Terms*—Software engineering, Explainable AI, Software defect prediction, Random Forest classifiers, XAI.**


## I. Introduction

Software systems are extremely complex and constantly evolving. Ensuring the smooth functioning of software program at some stage in and after deployment is required in Software projects. Creating dependable software program inside tight constraints of time, budget, and assets gives massive demanding situations. Predictive models are employed at some stage in the software program venture life-cycle , these model serve to assess development risk and facilitate the early identification [1]. Software defect prediction seeks to assume vulnerable areas of the software program at numerous life-cycle ranges, empowering improvement, testing, and management teams to gauge software program quality. These models allow engineers to focus on potential bugs, thereby improving software program quality and optimizing resource allocation.

Predicting defects is important in software engineering because of the resource-intensive nature of their detection and repair. This study examines the effectiveness of random forest classification in predicting software bugs using datasets from reputable repositories such as PROMISE, through the analysis of these datasets, provides insights into objective random forest classification and its application in software defect prevention. is to be found, thereby contributing to it.

Since AI is widely used in software systems , it's important to understand how machine machine learning (ML) makes decisions. However, some models are considered "black boxes" because they are not easily understood, making it difficult for users to understand identify factors that affect their decisions Obvious models are those cost forecasting serves to build reliability and encourage their use. In addition, it helps to understand, AI/ML models more descriptive and transparent and more understandable to stakeholders.

For this purpose, its essential to understand both local (individual) prediction and global prediction (i.e., model as a whole). Accordingly, aim to find answer for the research questions (RQ) below by using the model-agnostic explainability methods.

- **RQ1-** As per the model's assessment, which features from the dataset hold greater significance?
- **RQ2-** What role does each feature play in determining the outcome of individual predictions, and to what extent do they positively or negatively influence these predictions?
- **RQ3-** What is the overall influence of each feature on the model's predictions, and how does this contribute to the model's overall performance?
- **RQ4-** How do variations in feature values impact the likelihood of software defects across various versions?

## II. Related Work

As AI-based software systems grow more complex and opaque, individuals such as domain experts, system engineers, and customers often struggle to comprehend specific aspects of these systems. Conversely, there is a rising interest in the explainability of AI systems [2]. Explanations have many benefits, in enhancing system understanding and justifying decisions. Moreover, inadequate explanations breed distrust [3], diminish user acceptance and satisfaction, and hinders the adoption of new technologies. Thus, explanation is an critical part of the achievement of AI-based software program and have to be integrated throughout their development life cycle.

Exploring the impact of predictive decisions on software projects, an innovative Bootstrap aggregation-based hybrid-inducer ensemble learning (HIEL) technique was employed, utilizing a probabilistic weighted majority voting (PWMV) strategy.[4]

A study investigated the reliability of model-agnostic explanation techniques—such as LIME and Breakdown—when applied to various defect prediction models. The investigation found that these techniques produced varying results, raising doubts about their reliability in explaining defect prediction models[5].

The balance of model accuracy and reasonableness in predicting software bugs. A new superposed naive Bayes (SNB) classification model was introduced and its performance was evaluated in 13 global projects. Various comparisons and classifications were used, including ensemble learners, regression models, decision trees, support vector machines, Bayesian learners, neural networks, and rule-based learners. The findings showed that the SNB method achieved better accuracy and precision compared to the other algorithms tested[6].

Boosted random forest as a solution to the limitations of traditional multi-class classifiers due to bagging and feature selection effects. Experimental results reveal that our approach, with fewer decision trees, achieves superior speculation capability and enhances defect prediction accuracy, thus enhancing software quality [7][8][9].

A proposal was made to harness Explainable AI techniques to address the aforementioned challenges by rendering software defect prediction model predictions more practical, explainable, and actionable.[10]

Exploring the landscape of explainable AI (XAI) within the software engineering community, a thorough Systematic Literature Review was conducted, aiming to delve into the scope of research on XAI for Software Engineering.[11]

A comprehensive systematic review was conducted focusing on Explainable Artificial Intelligence (XAI) applied to Software Engineering. The aim was to identify and analyze the prevalent XAI techniques, tools, and evaluation criteria within the software engineering domain.[12]

## III. Methodology

### A. Methods Applied

- RFE: If we decide that an ML classifier is unreliable, the ubiquitous task is to modify the array of features and retrain the classifier to solve the generalization, which is called "Feature Selection". Explanation can help with this process by incorporating important features, especially by removing features that were not comprehensive.Recursive Feature Elimination (RFE) used to identify most important features within a dataset by iterative pruning of redundant features [13].
- LIME: A Local Interpretable Model-Agnostic Explanations (LIME) provides explanations for the predictions made by a machine learning model on individual instances. LIME operates by approximating the behavior of the underlying model within a local region around the instance of interest, allowing for the interpretation of complex black-box models. By generating interpretable explanations at the instance level, lime models offer insights into why a particular prediction was made, aiding in model transparency and trustworthiness. Fig 1 provides insight into the functioning of LIME.
- SHAP: Shapley value is a scheme that reflects on principles of game theory it is a model-agnostic

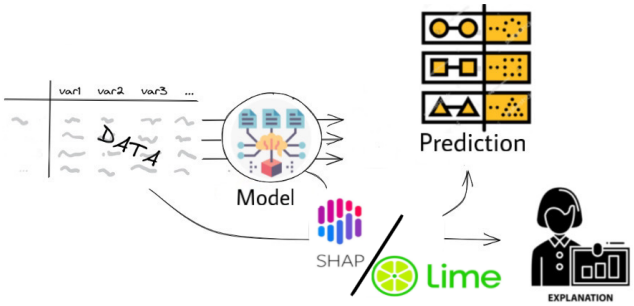


Fig. 1. LIME, SHAP providing insights for the predictions

technique since the method is devoid of dependence on black-box ML models. This method assigns each feature a value for prediction and it shows how much this feature represents the output of the model. Shapley values find out what would be considered a fair distribution between features in terms of their contributions towards outcomes by looking at all possible combinations with their orderings permutated. In Figure 2 we can observe an illustration that demonstrates SHAP's operational mechanics while also illustrating how it can be used to explain complex predictive modeling systems.

- PyExplainer: Understanding the features that influence the occurrence of defects is critical to effective software development and quality assurance. This study investigates the effects of varying attribute values on software error detection using advanced forecasting techniques.PyExplainer is a rule-based model-agnostic technique that utilizes a local rule-based regression model to learn the associations between the characteristics of the synthetic instances and the predictions from the black-box model[14]. Specifically, we use PyExplainer, a tool that enables real-time object support research by adjusting input values via sliders. By dynamically modifying attribute properties and controlling for changes in defect prediction results, we attempt to clarify the relationship between software-specific evaluation and efficiency.

TABLE I
DESCRIPTION OF SOFTWARE METRICS FEATURES

| Feature | Description |
|---|---|
| Version | The version number or identifier of the software. |
| Name | The name of the software component or class. |
| WMC (Weighted Methods per Class) | It represents the sum of the complexities of all methods in a class. It measures the size or complexity of a class. |
| DIT (Depth of Inheritance Tree) | It represents the depth of the inheritance tree for a class. It indicates how many levels of inheritance exist for a particular class. |
| NOC (Number of Children) | The number of immediate subclasses or children a class has. |
| CBO (Coupling Between Objects) | It measures the dependencies between different classes. A high CBO may indicate higher coupling between classes. |
| RFC (Response for a Class) | It represents the number of methods that can potentially be executed in response to a message received by an object of that class. |
| LCOM (Lack of Cohesion in Methods) | It measures the lack of cohesion among the methods of a class. A lower LCOM value indicates higher cohesion. |
| CA (Afferent Connections) | The number of classes that depend on the class. |
| CE (Efferent Connections) | The number of classes that the class depends on. |
| NPM (Number of Public Methods) | The total number of public methods in a class. |
| LCOM3 | Another measure of lack of cohesion, possibly a variant of LCOM. |
| LOC (Lines of Code) | The total number of lines of code in a class. |
| DAM (Data Access Metric) | It measures the number of attributes in a class that are accessed by methods outside the class. |
| MOA (Measure of Aggregation) | It measures the number of data members in a class that are of reference types. |
| MFA (Method-Field Access) | The ratio of the number of methods to the number of attributes in a class. |
| CAM (Cohesion Among Methods of Class) | It measures the degree of cohesion among the methods of a class. |
| IC (Inheritance Coupling) | It measures the degree to which a class is connected to its ancestor classes. |
| CBM (Coupling Between Methods) | It measures the number of method call connections between methods within a class. |
| AMC (Average Method Complexity) | The average complexity of methods in a class. |
| Bug (Error) | Bugs in a software |

### *B. Dataset Description*

The "ant-apache" dataset sourced from the PROMISE repository comprises software defect data extracted from multiple versions (1.3 to 1.7) of the Apache Ant project.Apache Ant is a popular Java-based build tool used for automating software build processes.The dataset was at the start curated and made to be had through the PROMISE repository, a properly-installed useful resource for empirical software engineering datasets.By analyzing how software program metrics and defects indicators evolve across those versions, we intent to benefit insights into the effect of model updates on software quality and defect proneness. Attribute information for ANT dataset is given in TABLE 1

### *C. Machine Learning Model*

In this research we chose the Random Forest classifier, which is a fast and known for its high efficiency. It is built on a decision tree algorithm, and searches for different

configurations of ratings, types, and device types by obtaining knowledge. Although there are many classifier detection algorithms to choose from, we chose Random Forest because of its superior overall performance. In particular, the random forest had the highest accuracy compared to the decision tree classifier, K Neighbours classifier, and MLP classifier.Moreover, the ensemble nature of a Random Forest that mixes multiple decision trees, Enhances its robustness and generalization talent.Thus assured the effectiveness of using random forest as a primary classifier and gaining knowledge on the liability.

In our rapidly advancing technological evolution, reliance on AI and software systems has expanded. However, with this increase comes a puzzle that as AI systems grow in complexity, the ability of individuals such as domain experts, engineers, end users and more to hear them internal performance decreases These challenges significantly hinder widespread adoption and trust in AI technologies.

## IV. Results

*A.* ***Results for RQ1-*** *As per the model's assessment, which features from the dataset hold greater significance?*

The aim of our study was to identify the most important factors according to the model, with particular emphasis on how the RFE affect the performance of a random forest model in different data types. Through this approach, we observed a consistent increase in model accuracy for each type of Ant data, indicating the importance of feature selection in simplifying model inputs. And reducing the dimension of the dataset to keep hold of the most informative features, RFE upon enhancing the model performance but also promotes efficiency in computation and interpretabilty of results. Feature selection contributes to model transparency allowing us to gain insights into the underlying factors that are driving predictions.

In this study we focused on the various version of Ant dataset, after applying Recursive Feature Elimination (RFE) to enhance the performance of a machine learning model on version 1.3. By implementing RFE, we identified a subset of 10 important features that increased the model's accuracy from 0.69 to 0.84. The 10 features are WMC, CBO, RFC, LCOM, LCOM3, LOC, CAM, AMC, MAX_CC, and AVG_CC, were determined to be the most influential for predicting outcomes in the dataset.

Applying the same RFE technique for all other versions of Ant dataset, we consistently observed better prediction by Random Forest model .Each iteration of feature selection tailored the model to the specific dataset version, highlighting the versatility and effectiveness of RFE in optimizing model performance.

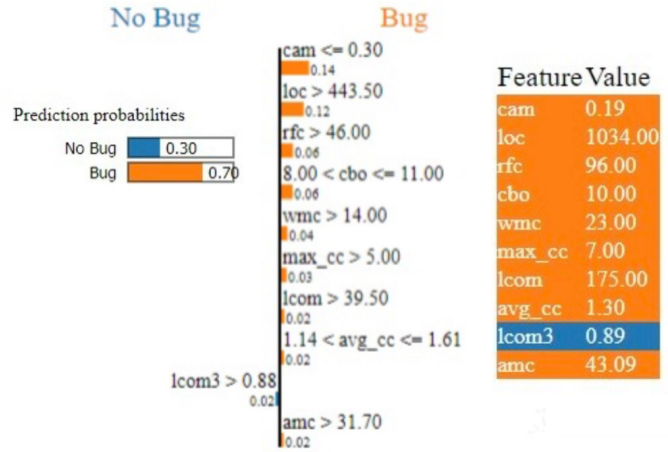


Fig. 2. LIME results for instance 6 of ANT 1.3 dataset

*B.* ***Result for RQ2-*** *What role does each feature play in determining the outcome of individual predictions, and to what extent do they positively or negatively influence these predictions?*

Up to here on research, we've found well known insights from a system gaining knowledge of model. Here after, we lookup on the local prediction, wreck down the version for a single instance prediction, and are searching for to understand the contribution of every function on an individual prediction. In order to classify a single instance of the dataset example as "yes" or "no", and explain that prediction by way of the machine learning model; We utilized LIME exclusively and analyzed the results obtained using this method.

To address the question of how the attributes of each feature contribute to the prediction results and their impact on model output based on the LIME explanation in row 6, We describe Fig 3 in detail as follows:

1) The model predicts a 70 percent probability of "Bug" (defective) and a 30 percent probability of "No Bug" (non-defective) for instance 6.
2) The color orange represents features that increase the prediction probability of the target class , while blue indicates features that decrease the prediction probability of the target class.
3) CAM <= 30: Lower values of CAM(coupling among methods) increases the prediction probability of target class by 14 percent
4) LOC > 443.50: Higher values of LOC(Lines of code) increases the prediction probability of target class by 12 percent

The central part of Fig 3 represents the predicted probabilities assigned by the model to each class (e.g., "Bug" and "No Bug"). This section provides insights into the model's confidence levels for different outcomes based on the local

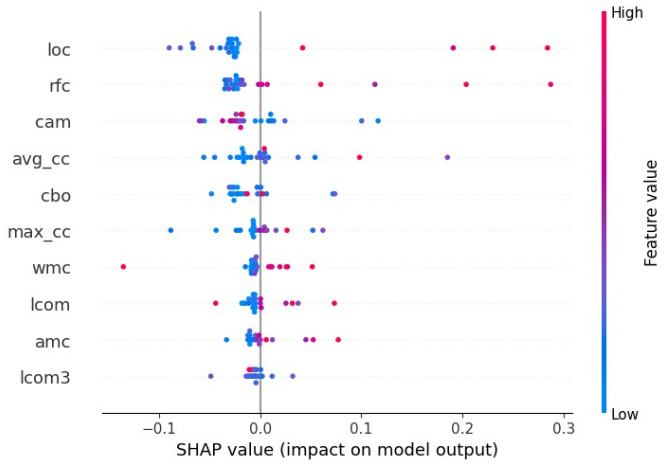


Fig. 3. SHAP summary plot for the dataset

neighborhood around the explained instance. The right side of the diagram displays the actual attribute values (features) of the explained instance, showcasing the specific input values that influenced the model's prediction. By examining this section, one can identify the most impactful features contributing to the model's decision, helping to interpret and understand the prediction outcome for the instance at hand.

*C.* ***Result for RQ3-****What is the overall influence of each feature on the model's predictions, and how does this contribute to the model's overall performance?*

In this research till here we have seen about the recursive feature elimination method in RQ1 and in RQ2 we have seen about the local prediction of LIME toward our dataset and how well the explainable AI model is performing. Finally, with RQ3, we have widened shap values and proposed to see how aggregating many SHAP values could give more detailed alternatives to permutation feature importance and partial dependence plots.

Wherein with recursive feature elimination in the research we have filtered out the most important features in our dataset, but still could get any insights about how a feature matters for the model with this methodology, to get the explanation we have used SHAP summary plot as shown in the Fig 4.

From Fig. 4 we get to know the detailed view about each feature's that affect the model prediction performance.While shade indicates each feature price as high or low, horizontal location indicates the SHAP values about whether or not the effect of that fee has triggered a better or decrease prediction performance. According to the plot, we look at that better values of overall LOC and RFC growth the version's prediction performance.

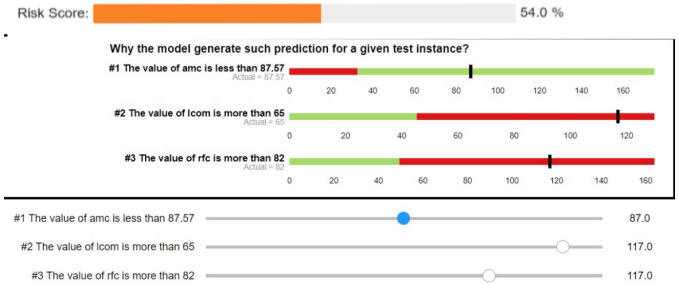


Fig. 4. Prediction of Risk score by PyExplainer

*D.* ***Result for RQ4-*** *How do variations in feature values impact the likelihood of software defects?*

To address the question of How do variations in feature values impact the likelihood of software defects? We describe Fig 5 in detail as follows:

In this research, we focused on investigating the impact of variables on the occurrence of software errors. Three specific factors were identified as particularly influential: Average Method Complexity (AMC), Lack of Cohesion in methods (LCOM), and Response for a Class (RFC). In our analysis of a single sample from the database, we found that by varying the values of these characteristics, the risk level for dust exposure reached 54 percent. This result indicates that variations in the values of AMC, LCOM and RFC have a significant effect on the occurrence of software errors, with most of the models showing a high risk of bugs.

## V. Conclusion

Using explainable AI methodologies is essential for improving transparency, trust with stake holders, and interpretability of machine learning models. Providing insights into model decision-making helps stakeholders recognize how particular features influence predictions, thereby facilitating informed decision-making and facilitating version deployment in critical applications.

Explanations by Local Interpretable Model-agnostic Explanations (LIME) and SHapley AdditiveexPlanations (SHAP) indicate the importance of two different XAI techniques. Due to methods commonly used such as LIME (local interpretable model-agnostic explanations) is designed to explain very complex models at the level of single points. The transparent and debuggable nature of this model is conveyed through understanding why it gave the prediction for certain data points, hence increasing the transparency and reliability of the model. On the opposite side, SHAP provides global explanations by assigning feature importance values to explain the behavior of a model across the dataset. It can be used to gain insights into what drives predictions at scale by

showing how features collectively affect them.

What sets PyExplainer apart from other methods in XAI is that it allows interactive exploration of explanation of models using sliders that adjust feature values.[15] Most often than not, LIME and SHAP usually provide explanations which are static but this isn't true with PyExplainer as users can change values dynamically while observing changes in real time to their models' predictions . The involvement of an interface where one can interact gives better understanding on importance of features hence this makes PyExplainer unique among other XAI techniques since it could also be used for learning purposes ,model checking or scenario analysis.

## REFERENCES


[1] Ozak and A. Tarhan, "Early software defect prediction: A systematic map and review," vol. 144, no. May, pp. 216-239, 2018.

[2] Z. C. Lipton, "The of Model The Interpretability," pp. 1–28, 2018 28.

[3] M. Lahijanian and M. Kwiatkowska, "Social Trust: A Major Challenge for the Future of Autonomous Systems," pp. 189-194, 2016.

[4] B. Umamaheswara Sharma and R.Sadam, (2020). A novel ensemble learning model for accurate software defect prediction. Journal of Systems and Software, 165, 110596.

[5] J. Shin, R. Aleithan, J. Nam, J. Wang, and S. Wang, "Explainable Software Defect Prediction: Are We There Yet ?," pp. 1-12, 2021.

[6] T. Mori and N. Uchihira, Balancing the trade-off between accuracy and interpretability in software defect prediction. Empirical Software Engineering, 2019.

[7] K Dinesh Kumar et al., "Attribute-oriented Classification with Variable Importance using Random Forest Model", International Journal of Recent Technology and Engineering, ISSN: 2277-3878, Vol. 8, No. 2S3, pp. 1630-1635, 2019.

[8] R. Prasanna Kumar, "Sequential feature selection for heart disease detection using random forest", Iraqi Journal of Science, 2021.

[9] Thulasi Bikku,K. P. N. V. Satyasree "A Boosted Random Forest Algorithm for Automated Bug Classification ", Lecture Notes in Networks and Systems book series (LNNS,volume 650).

[10] C. K. Tantithamthavorn and J. Jiarpakdee, (2021). Explainable artificial intelligence for software defect prediction: A practical roadmap. Journal of Systems and Software, 175, 110928.

[11] A. M. Khani and N. S. Bommi (2023). Systematic literature review of explainable artificial intelligence for software engineering. Information and Software Technology, 142, 106771.

[12] G. Vilone and L. Longo (2022). A systematic review on explainable artificial intelligence applied to software engineering. Journal of Software: Evolution and Process, 34(1), e2319.

[13] M. Awad and S. Fraihat, Recursive Feature Elimination with Cross-Validation with Decision Tree: Feature Selection Method for Machine Learning-Based Intrusion Detection Systems. J. Sens. Actuator Netw. 2023, 12, 67.

[14] C. Pornprasit, C. Tantithamthavorn, J. Jiarpakdee, M. Fu and P. Thongtanunam, "PyExplainer: Explaining the Predictions of Just-In-Time Defect Models," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021.

[15] Mredulraj S. Pandianchery, Gopalakrishnan E. A, Sowmya V, Vinayakumar Ravi, and Soman K. P, "Explainable AI Framework for COVID-19 Prediction in Different Provinces of India," arXiv:2201.06997 [cs.CY], 2022.



**Balaji V** is a B.Tech student in Computer Science and Engineering from the 2020-24 batch at Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.



**Kennith Ronnie E** is a B.Tech student in Computer Science and Engineering from the 2020-24 batch at Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.

**Sailesh G** is a B.Tech student in Computer Science and Engineering from the 2020-24 batch at Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.

**Prasanna M** is a B.Tech student in Computer Science and Engineering from the 2020-24 batch at Amrita School of Computing, Coimbatore, Amrita Vishwa Vidyapeetham, India.

**Sabarish B A** currently serves as Assistant Professor (Sl. Gd.) at the School of Computing, Amrita Vishwa Vidyapeetham, Coimbatore Campus. B.A.Sabarish received a Ph.D. degree under Faculty of Engineering, from Amrita Vishwa Vidyapeetham, Coimbatore in 2022. His areas of research include Machine Learning, Artificial Intelligence, Spatial databases and Software Engineering.