

AN EXPLAINABLE MACHINE LEARNING FRAMEWORK FOR EARLY SOFTWARE DEFECT PREDICTION: ENHANCING SOFTWARE QUALITY THROUGH INTELLIGENT CLASSIFICATION

Muhammad Zulqarnain Siddiqui^{*1}, Hamza Bin Irfan², Muhammad Bilal Mansoor³

^{*1}Associate Professor, Department of Computer Science, Iqra University, Karachi,

²Department of Computer Science and Information Technology, NED University of Science & Technology, Senior Consultant - Cloud, Data, and AI, Software Wing, ABATS - Ali Bin Ali Technology Solutions, Doha, Qatar.

³Department of Computer Science and Information Technology, NED University of Science & Technology, Senior Consultant - Mobile App Developer, Cloud and Mobility Department, Systems Limited.

¹zulqarnain@iqra.edu.pk, ²hamza.irfan@abats.com.qa, ³mbilal.mansoor@systems ltd.com

DOI: <https://doi.org/10.5281/zenodo.21847702>

Keywords

software defect prediction; explainable artificial intelligence; XGBoost; SHAP; software metrics; class imbalance; software quality assurance

Article History

Received: 11 April 2026

Accepted: 15 May 2026

Published: 31 May 2026

Copyright @Author

Corresponding Author: *

Muhammad Zulqarnain Siddiqui

Abstract

Software defect prediction can support early allocation of verification and testing effort by identifying modules that are likely to contain defects before release. This study develops a leakage-safe explainable machine learning framework and evaluates it on four cleaned NASA Metrics Data Program benchmark datasets: CM1, JM1, KC1, and PC1. Within each training fold, the framework applies median imputation, correlation-aware feature filtering, robust scaling, and Synthetic Minority Over-sampling Technique balancing. Random Forest, radial-basis-function Support Vector Machine, XGBoost, and LightGBM are compared using stratified five-fold cross-validation and six complementary metrics. Across projects, XGBoost provided the strongest overall balance, attaining the highest macro-averaged accuracy (0.816), precision (0.435), and F1-score (0.402). Random Forest achieved the highest ROCAUC (0.761), LightGBM achieved the highest precision-recall AUC (0.429), and Support Vector Machine achieved the highest recall (0.470), demonstrating that the preferred model depends on the operational cost of missed defects and false alarms. A Friedman test detected an overall model difference for ROCAUC (chi-square = 8.40, $p = 0.038$), whereas differences for the remaining metrics were not statistically significant across the four projects. SHAP analysis of XGBoost on JM1 identified blank lines, numbers of unique operators and operands, design complexity, Halstead difficulty, cyclomatic complexity, and total lines of code as influential predictors. The framework therefore combines competitive prediction with transparent global and module-level explanations, while avoiding optimistic leakage from preprocessing or class balancing. The results support explainable defect prediction as a decision-support mechanism for risk-based testing rather than as an automatic replacement for engineering judgment.

1. INTRODUCTION

Late detection of defects in the software life cycle can result in multiple cycles of debugging, regression testing, redeployment, and maintenance. Software defect prediction is the practice of learning the relationship between

historic measurements of software and its defect labels in order to estimate the likelihood of defects in new software modules. A working model could assist quality assurance teams to determine which code should be inspected and tested given limited time and resources. Static

software metrics have an advantage in being easily calculated even before deployment without execution traces.

Past research on defect prediction has found that code attribute-based models can be used for this task, although they exhibit sensitivity to data quality, class imbalance, metric correlation, experimental methodology, and classifier tuning [1-3]. Defective modules are normally in the minority. Therefore, a model can achieve high accuracy simply by predicting non-defective modules for the majority of cases while failing to identify defective modules. The performance measures of accuracy need to be supplemented by measures of precision, recall, F1-score, ROC-AUC, and Precision-Recall AUC.

The second challenge relates to interpretability. While it is obvious whether a probability or a binary label was returned by a module classifier, this alone does not say why this decision was made. Such an explanation is crucial when it comes to differentiating between real engineering signals and unstable artifacts, explaining results to practitioners, and drawing attention to those metrics which played role in forming a certain prediction. Explaining artificial intelligence provides global insights about model behavior and local explanations for particular modules, but explanations have to be produced from a well-validated predictive pipeline [14-17].

The current work suggests and empirically evaluates a full-fledged framework comprising of leakage-safe pre-processing, correlation-driven feature selection, class re-balancing, comparative classification and SHAP-based explanation. Four pre-processed NASA Metrics Data Program datasets are used in order to maintain comparability with the existing defect-prediction literature and mitigate known data quality issues [3]. The goal of the study is not to establish supremacy of a particular algorithm. On the contrary, the focus is on exploring the trade-offs of various models and on producing explanations for predictions.

The main contributions include the following:

An imputation-robust pipeline where imputation, feature filtering, normalization, and SMOTE are calculated solely based on each training fold.

A fair comparison between Random Forest, Support Vector Machine, XGBoost, and LightGBM on four benchmark projects after cleaning, based on six different metrics.

A statistically conservative cross-project analysis that discerns the strength of individual metrics without the assumption of universal supremacy. SHAP-based explanations of the global and local importance of the features in an XGBoost classifier that performs well.

An executable analysis script along with data and model descriptions.

2. Related Work

2.1 Software Metrics and Defect Prediction

Static Defect Prediction Studies normally involve the use of size, complexity and Halstead-like features. The McCabe cyclomatic complexity feature represents an estimation of the number of linearly independent paths of execution [4] while Halstead features describe program vocabulary, length, volume, difficulty and effort based on operator and operand count [5]. Features such as lines of code and comments indicate the module size and document features respectively. Such features are indicators but not direct causes of defects and yet they have the advantage of easy accessibility and lower costs of acquisition.

Menzies et al. established that code features could be used to build good defect predictors while Lessmann et al. established that conclusions drawn could be contingent upon the design of benchmark used and also the set of classifiers considered [1,2]. The importance of data quality assessment before any modelling was subsequently stressed. Shepperd et al. revealed that there were duplicates and inconsistencies in the original NASA data set and also provided cleaned data sets that are suitable for reproducible empirical research [3]. This study uses the cleaned D-double prime data sets.

2.2 Imbalance, Correlated Metrics, and Evaluation Bias

Defect datasets are usually imbalanced due to the preponderance of non-defective modules. Rebalancing could help with sensitivity towards the minority class, although its impact differs for different classifiers and can alter the meaning of the model [6]. SMOTE creates synthetic minority examples by interpolation of nearby examples and is commonly used as a way of re-sampling instead of replicating [13]. It is important to note that the re-sampling procedure should be done post train/test split and using only training data. Otherwise, it will lead to the information leakage

from validation data into the training distribution.

Software metrics show considerable redundancy: for instance, several Halstead metrics are linearly dependent, and measures of size are highly correlated with complexity metrics. The high correlation between the variables destabilizes the model interpretation without decreasing the predictive power of the model [7,8]. The proposed approach uses transparent Spearman correlation criterion on each fold of training data to decrease redundancy prior to explanation.

2.3 Explainable Defect Prediction

While the random forest and gradient boosting algorithms are able to model non-linear interactions, they lack a clear representation based on coefficients. SHAP expresses a prediction in terms of contributions of each feature with respect to the expected output of the model [14]. Efficient TreeSHAP methods make this possible for tree ensembles and allow aggregation to compute the global feature importance [15].

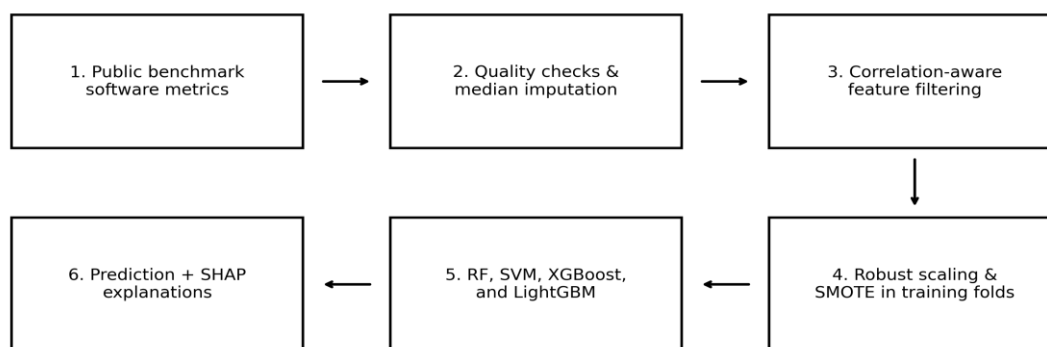
Previous work done on defect prediction has studied model-agnostic explanation methods and

found that such methods were valued by people, but there is the possibility that different models yield different explanations [16,17]. This leads to two considerations in this study. First, explanations are created only after evaluation of the model is completed. Second, the explanation is presented as model-specific associations and not as an indication that changing metrics would fix the issues.

3. Proposed Explainable Machine Learning Framework

The described framework has six interconnected steps: acquisition of benchmarks, data quality control, training folds pre-processing, correlation-based features filtering, class-balanced models creation and SHAP explanations. Step-by-step process is shown on Fig. 1. Each data-driven transformation is re-trained separately for each training fold during cross-validation. Held-out fold is transformed based on the parameters obtained from the training fold but never participates in median selection, variable selection, scaling and synthetic samples generation.

Leakage-safe explainable defect-prediction workflow



Stratified 5-fold cross-validation: every preprocessing step and SMOTE are refitted inside the training fold

Figure 1. Leakage-safe workflow for explainable software defect prediction.

There are two types of outputs from the framework. First, the predictive output, which includes either the defect probability or decision score along with the binary class. The second output is the explanatory output, which includes the following three components: (a) Global ranking through the average absolute value of

SHAP values; (b) Plot indicating the effect of high/low feature values on the prediction; and (c) Waterfall explanation of a selected module. It is important to note that the decoupling of prediction and explanation is deliberate.

4. Materials and Methods

4.1 Datasets and Outcome Definition

The analysis of the data set used the CM1, JM1, KC1, and PC1 from the D" collection of the NASA Metrics Data Program that is kept for empirical defect prediction research work. Each entry in the data set refers to a software component; each predictor is a static software

measure, and the dependent variable indicates whether the software module was classified as defective ($Y=1$) or non-defective ($N=0$). There are altogether 9,888 software modules in the four data sets, out of which 2,003 are defective; however, performance measures were first computed per project.

Table 1. Dataset composition and retained feature counts.

Dataset	Modules	Original metrics	Defective	Non-defective	Defect rate	Metrics retained per fold
CM1	327	37	42	285	12.8%	17-19
JM1	7720	21	1612	6108	20.9%	10
KC1	1162	21	294	868	25.3%	7
PC1	679	37	55	624	8.1%	16-17

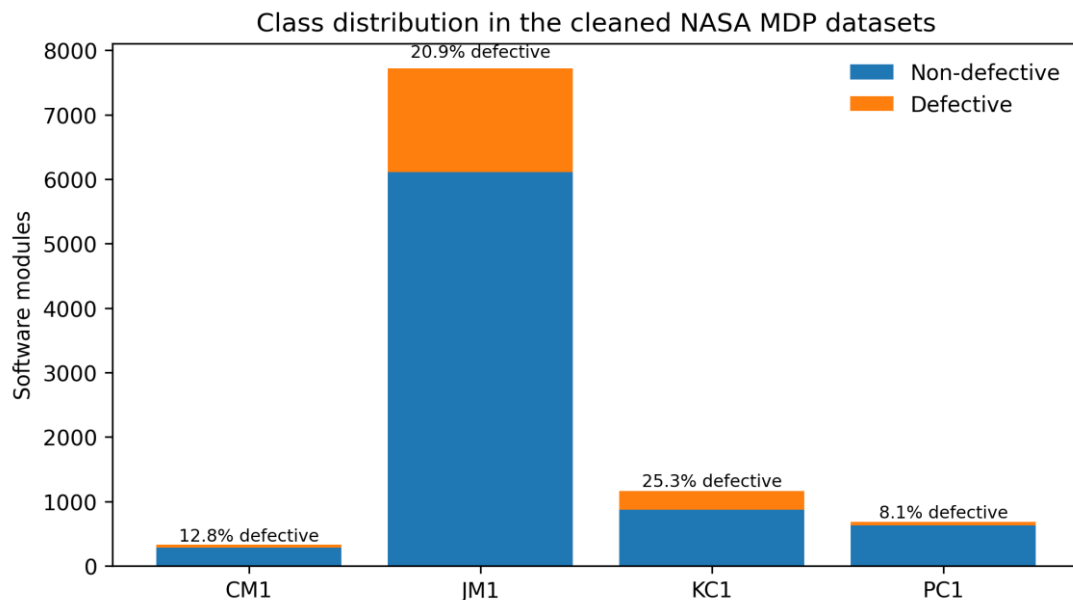


Figure 2. Class distribution in the four cleaned NASA MDP datasets.

The rate of defects varies from 8.1% for PC1 to 25.3% for KC1. Such differences suggest that one single performance metric may not be enough. For instance, accuracy can be easily manipulated in PC1 but not in KC1 because precision-recall AUC captures the true performance given class prevalence [19].

4.2 Static Software Metrics

Candidate predictor variables include metrics based on lines of code, comment and blank line counts, branch counts, McCabe complexity metrics, and Halstead metrics based on operator and operand counts. The number of metrics

used varies between projects; CM1 and PC1 utilize 37 candidate metrics while JM1 and KC1 use only 21. Due to the high degree of correlation between some of these metrics (as they describe similar characteristics of module size and complexity), feature reduction is an important part of the proposed framework.

4.3 Leakage-Safe Preprocessing and Feature Filtering

Stratified 5-fold cross-validation was performed with shuffling and seed 2026. Stratification ensured that the fraction of defectiveness specific

to the project was preserved for each fold. Each split was completed as follows:

1. Missing values for predictors were imputed using the median from the training fold.
2. Constant metrics were filtered out. Next, the absolute Spearman correlation between the two non-constant metrics was calculated based on the training fold metrics. If the absolute value of Spearman correlation for a pair exceeded 0.90, then the metric which had a weaker absolute Spearman correlation to the outcome in the training fold was removed; deterministic column ordering resolved ties.
3. The remaining metrics were scaled using robust scaler, based on the median and interquartile range calculated from the training data.
4. SMOTE was performed for the training observations only, with 3 nearest neighbors. The validation fold had the natural class distribution.
5. The model was fit using the balanced training set and evaluated on the untouched validation fold.

The number of features could thus change across folds, depending on whether or not the absolute Spearman correlation close to the threshold varied due to changes in the training sample.

4.4 Classification Algorithms

Four classifiers that use supervision have been chosen since they demonstrate complementary learning algorithms and are often used as baselines in software analytics. Random Forest learns an average of decorrelated decision trees and is fairly insensitive to nonlinear relationships [9]. Radial Basis Function Support Vector Machine learns a nonlinear margin in a higher-dimensional feature space [10]. XGBoost learns regularized gradient boosting in an incremental way [11], and LightGBM uses the efficient leaf-wise gradient-boosting approach [12].

The hyperparameters were set prior to performing cross-validation as opposed to being tuned on evaluation folds. This allows a straightforward comparison but does not claim to find the optimal configuration for each project, even though it avoids any kind of optimization leakage. The hyperparameters are detailed in Appendix A and the provided script for analysis..

4.5 Evaluation Measures

TP, FP, TN, and FN represent true positive, false positive, true negative, and false negative respectively. The accuracy metric describes how many predictions have been correctly made. Precision captures the ratio of predicted defectives which are indeed defective, while recall captures the ratio of actual defectives identified. The F1 score is the harmonic mean of precision and recall. The ROC AUC score captures the performance of ranking, while the precision-recall AUC score gives more weight to the minority class. All the above scores were calculated since none captured the entire trade-off.

$$\text{Precision} = TP/(TP + FP); \quad \text{Recall} = TP/(TP + FN); \\ F1 = 2(\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall})$$

Results at the fold level were averaged per project and per model. Macros at the cross-project level were computed as the arithmetic average, giving each project equal weight. Statistical significance was tested using the Friedman test on project-level data for each individual metric with models considered as repeated measures and projects as blocks [18]. With only four projects available, p-values and rankings were cautiously interpreted without making any broad claim about superiority.

4.6 Explainability Analysis

XGBoost was chosen for explanation due to its better macroscopic performance in terms of accuracy, precision, and F1 score and because of the efficient computation of SHAP values specific to each tree. The stratified 80/20 split of JM1 dataset was created independently with the same preprocessing process used only during training phase. The SHAP values of 600 test modules were calculated. The mean absolute SHAP values were used to quantify the global effect, the beeswarm plot showed the direction and distribution of SHAP values, and the waterfall plot gave an explanation of one correct high-risk module detection.

5. Results

5.1 Project-Level Predictive Performance

The table below provides the mean of the five-fold validation results for each data model pair. The performance is highly project dependent. XGBoost performed best in terms of F1 score in CM1, while SVM performed best in JM1 and KC1, whereas LightGBM performed best in

PC1. For the ROC-AUC metric, Random Forest performed best in JM1, KC1, and PC1, while

XGBoost was the best for CM1.

Table 2. Five-fold cross-validation means by project and model.

Dataset	Model	Accuracy	Precision	Recall	F1	ROC-AUC	PR-AUC
CM1	LightGBM	0.832	0.287	0.233	0.251	0.801	0.421
CM1	Random Forest	0.835	0.360	0.308	0.324	0.785	0.373
CM1	SVM	0.820	0.379	0.378	0.358	0.724	0.350
CM1	XGBoost	0.856	0.433	0.378	0.399	0.806	0.427
JM1	LightGBM	0.779	0.457	0.307	0.367	0.700	0.405
JM1	Random Forest	0.780	0.466	0.349	0.399	0.717	0.423
JM1	SVM	0.715	0.363	0.476	0.411	0.685	0.367
JM1	XGBoost	0.764	0.424	0.368	0.394	0.700	0.401
KC1	LightGBM	0.741	0.485	0.415	0.446	0.689	0.476
KC1	Random Forest	0.737	0.478	0.425	0.449	0.693	0.470
KC1	SVM	0.708	0.435	0.497	0.463	0.673	0.437
KC1	XGBoost	0.743	0.490	0.408	0.445	0.691	0.480
PC1	LightGBM	0.910	0.448	0.364	0.395	0.830	0.415
PC1	Random Forest	0.898	0.409	0.345	0.369	0.851	0.421
PC1	SVM	0.813	0.221	0.527	0.311	0.749	0.281
PC1	XGBoost	0.900	0.392	0.364	0.372	0.836	0.395

5.2 Cross-Project Comparison

The method of macro averaging makes sure that every project contributes equally. The best macro accuracy was achieved by XGBoost, with a score of 0.816, while its precision was 0.435 and F1-

Score was 0.402. The highest ROC-AUC was scored by Random Forest, which had a score of 0.761. LightGBM had the best PR-AUC, scoring 0.429, while SVM had the highest recall, scoring 0.470.

Table 3. Macro performance across four projects (mean +/- standard deviation).

Model	Accuracy		Precision		Recall		F1		ROC-AUC		PR-AUC	
LightGBM	0.816	+/- 0.073	0.419	+/- 0.090	0.330	+/- 0.078	0.365	+/- 0.083	0.755	+/- 0.071	0.429	+/- 0.032
Random Forest	0.813	+/- 0.070	0.428	+/- 0.055	0.357	+/- 0.049	0.385	+/- 0.052	0.761	+/- 0.071	0.421	+/- 0.040
SVM	0.764	+/- 0.061	0.349	+/- 0.091	0.470	+/- 0.065	0.386	+/- 0.066	0.708	+/- 0.035	0.359	+/- 0.064
XGBoost	0.816	+/- 0.075	0.435	+/- 0.041	0.379	+/- 0.020	0.402	+/- 0.031	0.758	+/- 0.074	0.425	+/- 0.039

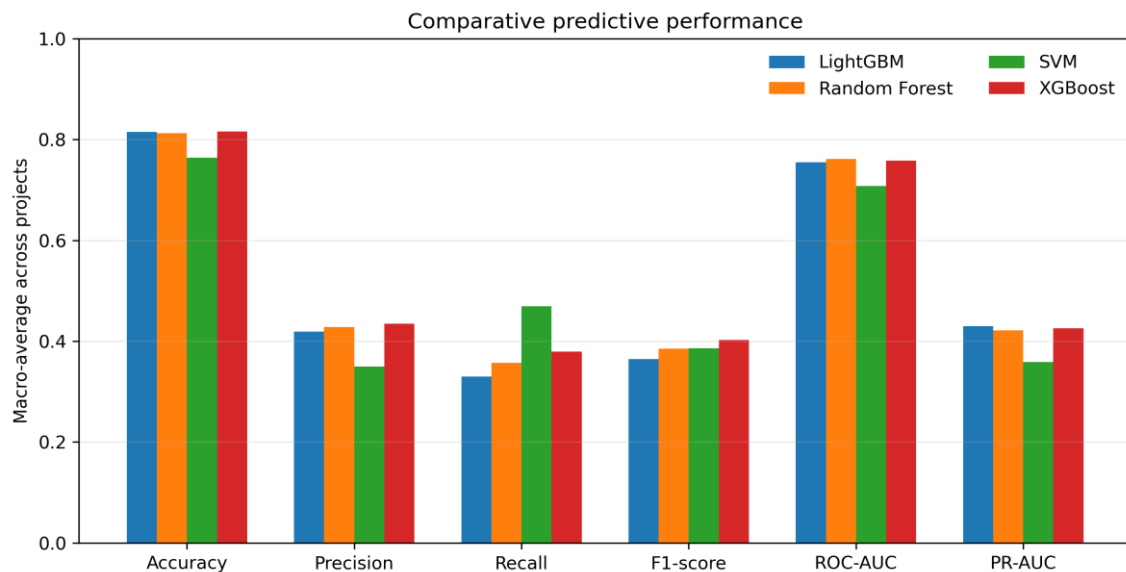


Figure 3. Macro-averaged predictive performance across the four projects.

The smallest differences occur among the tree ensembles. XGBoost, LightGBM, and Random Forest have macro accuracies within approximately 0.003 of one another, so the practical model choice should be guided by the preferred error profile and explanation

requirements. The absolute F1 and PR-AUC values are moderate, reflecting the difficulty and imbalance of the cleaned benchmark projects rather than the near-perfect results sometimes reported under less stringent validation.

5.3 Statistical Comparison

Table 4. Friedman tests across project-level model results.

Metric	Friedman chi-square	p-value	Interpretation at alpha = 0.05
Accuracy	7.500	0.058	Not significant
Precision	3.300	0.348	Not significant
Recall	7.342	0.062	Not significant
F1-score	1.200	0.753	Not significant
ROC-AUC	8.400	0.038	Significant
PR-AUC	7.200	0.066	Not significant

The only metric showing statistical difference in overall performance between the models was ROC-AUC (chi-square = 8.40, $p = 0.038$). The mean ROC-AUC scores were 1.50 for Random Forest, 2.00 for XGBoost, 2.50 for LightGBM, and 4.00 for SVM. The accuracy, recall, and PR-AUC values were close to, but below, 0.05. With the few blocks of projects in the experiment, the statistical test is underpowered, so it should be taken as an indicator of tendencies, not ranking of the algorithms.

5.4 JM1 ROC Analysis

Figure 4 depicts the out-of-fold ROC curves for JM1. In terms of ROC-AUC, we have the following order of models: Random Forest with ROC-AUC of 0.717; LightGBM with ROC-AUC of 0.700; XGBoost with ROC-AUC of 0.699; and SVM with ROC-AUC of 0.684. These curves are quite similar at all the thresholds. This supports the idea that small differences numerically should not be over interpreted.

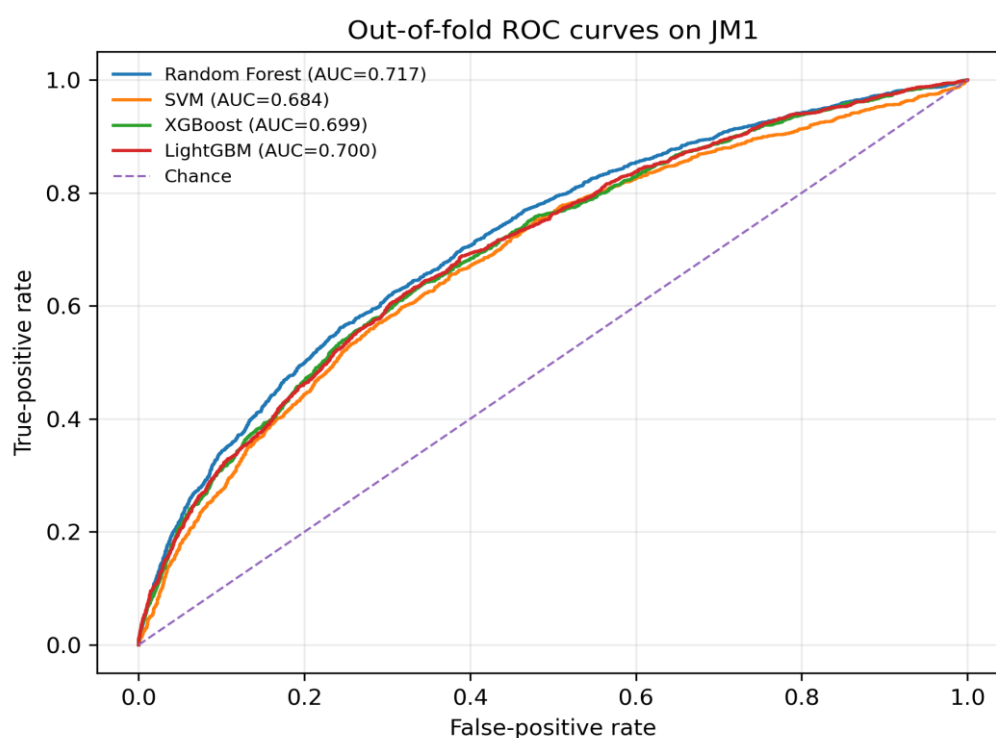


Figure 4. Out-of-fold ROC curves for the four classifiers on JM1.

5.5 Global Explainability

Explanations of XGBoost for the global data on JM1 are shown in Figures 5 and 6. The variable having the highest average absolute SHAP value was LOC_BLANK with a value of 0.518, much

higher than any other metric. Number of unique operators and operands, design complexity, Halstead difficulty, cyclomatic complexity, and number of lines of code were the other important variables.

Table 5. Global XGBoost SHAP importance on the JM1 explanation set.

Rank	Metric	Mean absolute SHAP value
1	LOC_BLANK	0.518
2	NUM_UNIQUE_OPERATORS	0.176
3	NUM_UNIQUE_OPERANDS	0.149
4	DESIGN_COMPLEXITY	0.137
5	HALSTEAD_DIFFICULTY	0.113
6	CYCLOMATIC_COMPLEXITY	0.112
7	LOC_TOTAL	0.107
8	LOC_COMMENTS	0.096
9	LOC_CODE_AND_COMMENT	0.077
10	ESSENTIAL_COMPLEXITY	0.037

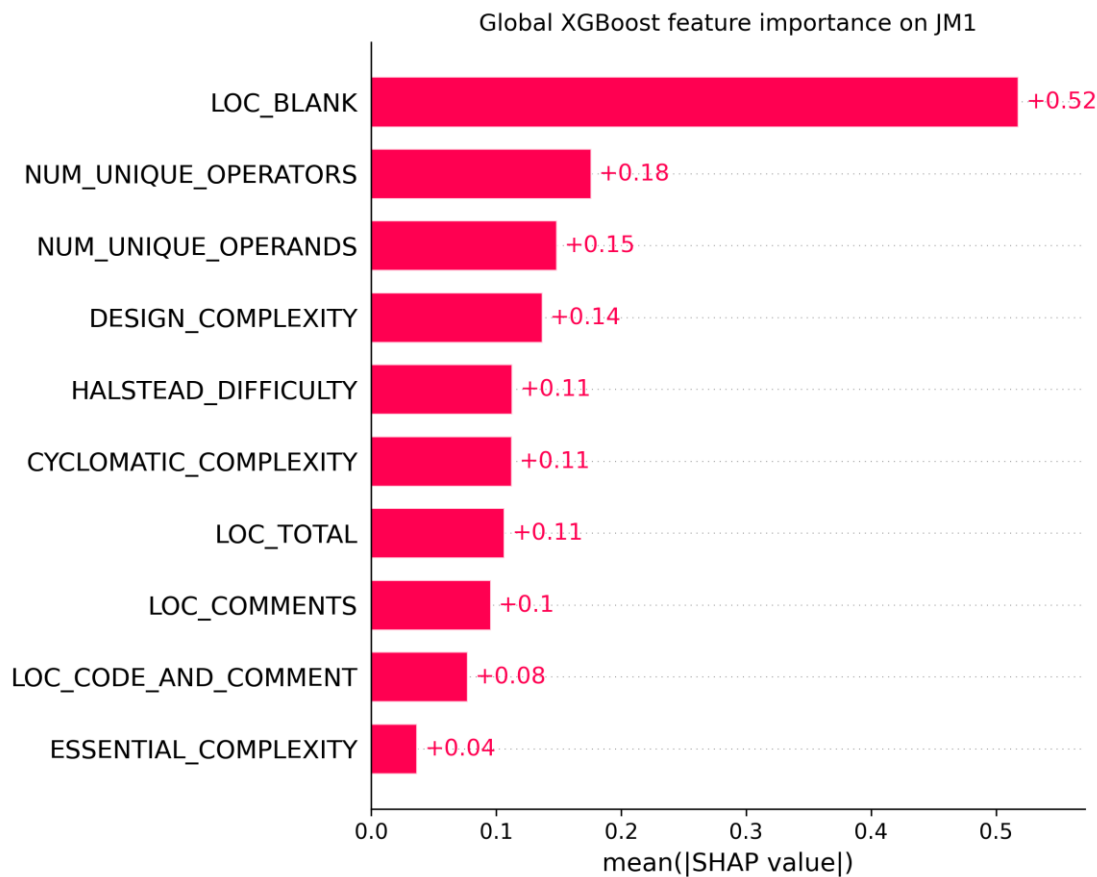


Figure 5. Mean absolute SHAP importance for the JM1 XGBoost model.

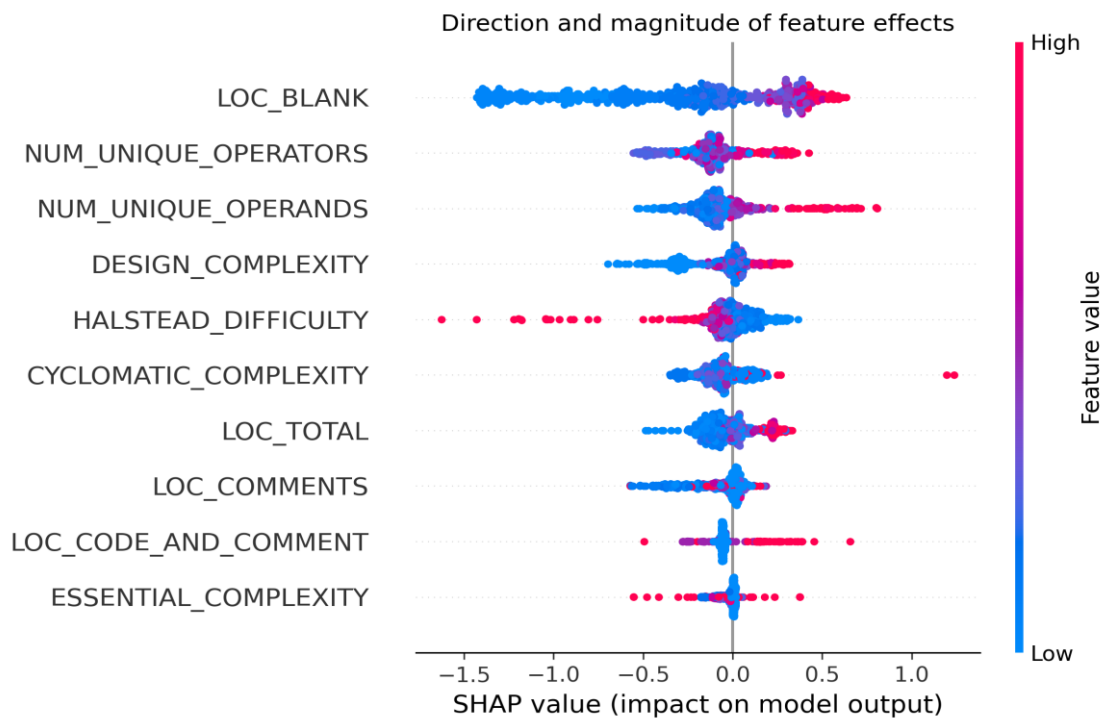


Figure 6. Direction and magnitude of feature effects in the JM1 XGBoost model.

Directionality is incorporated into the ranking by the beeswarm plot. For instance, high scores of a

number of size and complexity features usually push predictions to a higher chance of defect,

though the relationship is not necessarily monotonic with respect to all the features. The distribution of the SHAP values highlights interactions and heterogeneities between modules. The correlated features were removed before training the model, thus the explanations are more cohesive, yet still model- and data-dependent.

5.6 Local Explanation of a High-Risk Module

As shown in Figure 7, the following is the explanation of a properly classified defective JM1

with a predicted probability of 0.962. Higher cyclomatic complexity, number of unique operands, blank lines, and code & comment lines contributed to the higher prediction. Halstead difficulty and essential complexity decreased the output of the model for this module. This clearly indicates that a single prediction may be based on several features that compete for each other's value contribution. The explanation can help in reviewing the score based on recorded values.

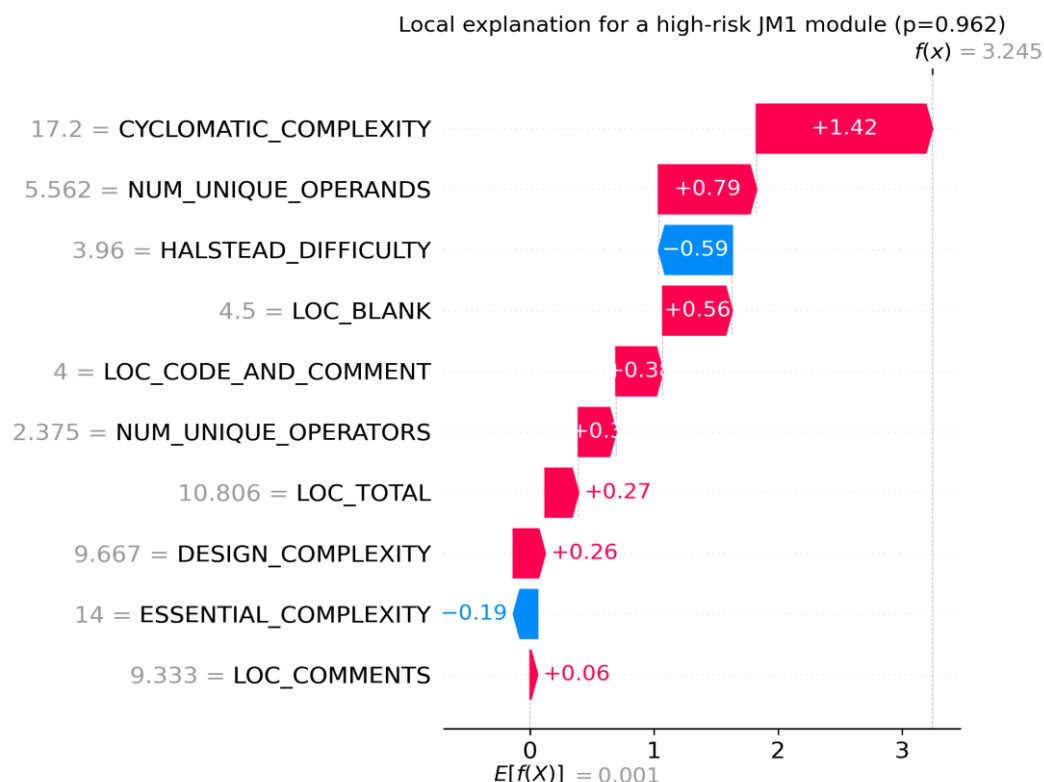


Figure 7. Local TreeSHAP explanation for a high-risk JM1 module (predicted probability = 0.962).

6. Discussion

6.1 Interpretation of Comparative Results

However, the results do not lend credence to the notion that any one classifier reigns supreme over all defect prediction tasks. XGBoost had the best overall tradeoff and therefore would serve as a sensible default in situations where both precision and F1 score are desired. The best classifier in terms of ROC-AUC is the Random Forest classifier while the LightGBM had a slight edge in PR-AUC. The SVM performed the best at generating high recall scores, which might suit well in cases where false negatives have greater consequences than false positives.

The ranking of the models also varies from project to project, which is quite natural since the data sets differ in feature set, class imbalance, project nature, and sample size. For instance, PC1 has 55 defective modules, meaning that there will be higher variability in fold-wise precision and recall rates. Thus, deployment of the chosen model needs to be based on the local project experience as opposed to adopting the winner blindly.

The moderate performance scores represent a strong point in the evaluation procedure but not a weakness in the framework. Preprocessing and oversampling took place only within training folds, clean datasets were used, and performance

was evaluated on naturally imbalanced validation folds. It minimizes the probability of the optimistic estimates appearing due to the use of synthetic instances, global feature selection, and duplicating examples across the evaluation boundary.

6.2 Implications for Software Quality Practice

A defect prediction model should be used as part of a triage workflow. The following procedure is recommended:

1. Calculate the static metrics that are needed for the model, once the module or release candidate is ready.
 2. Provide a defect risk score and sort modules, instead of assigning a simple binary label.
 3. Choose an appropriate decision threshold that reflects inspection capability and the costs of false negatives.
 4. Periodically examine the global explanation to see if the model makes use of technically reasonable signals.
 5. Provide a local explanation of each risky module so that the reviewer can see what factors led to its selection.
 6. Keep track of review results and re-train the model if project practices, programming languages, or defects criteria are changing.
- SHAP-based explanations may help to communicate between data scientists and software engineers better. The reviewer can determine whether the score is determined mainly by size, control flow complexity, operator set, or some weird combination of metrics. An explanation can also point to potentially harmful dependencies on the metric that has been modified due to tool changes..

6.3 Explainability and Actionability

Global importance can help facilitate discussions about processes, but it should not be directly translated into prescriptive guidelines. A large value of SHAP in the global sense for the LOC_BLANK variable implies only that the model often utilizes this variable in the framework of JM1 data set; it cannot prove that blank lines result in defects. This variable is related to programming style, generated code, module structure or something else.

Explanations on a local level are more practical when it comes to prioritizing actions since such explanations will explain what makes a specific module different from the baseline of the model. Nonetheless, the stability of explanations has to

be taken into account. The literature shows that rankings of explanations can be dependent on the sampling and model parameters [16,17].

7. Threats to Validity

7.1 Construct Validity

The binary defect categorization relies on the procedure used in collecting the data and linking it to the defect. The static metrics account for measurable qualities of the code but do not account for all the reasons why a defect could arise, like the experience of the developers, requirements changes, or architectural dependencies.

7.2 Internal Validity

Leakage is the primary source of internal validity problems. Leakage was avoided by applying imputation, feature selection, normalization, and SMOTE only on the training set. However, the correlation filter makes its decision using training labels and hence it is a supervised filter. The estimates made by the filter may differ in small samples. Random seeds ensure replicability but results may change with different thresholds.

7.3 Conclusion Validity

A total of only four projects was considered in the Friedman test across projects, thus reducing its statistical power. Various measures were considered, and the single p-value that is less than 0.05 needs to be taken with caution. The standard deviations are for variation among projects for the macro analysis and variation among folds for the project-level analysis and not confidence intervals for future projects.

7.4 External Validity

The cleaned NASA projects are widely used but are historical and may not reflect modern development languages, continuous integration, microservices, or contemporary code-review practices. The results should be replicated on current industrial and open-source projects, preferably with temporal validation. Performance can also change when the defect definition, metric tool, unit of analysis, or release process changes.

7.5 Explainability Validity

The SHAP values correctly decompose the selected model using the selected background distribution; however, they do not create any

causal relationship nor can they guarantee explanation stability with respect to retraining. The correlated feature filtering eliminates the possibility of one ambiguity; however, unmeasured confounders and model interaction ambiguities still persist.

8. Conclusion

This study introduced a leakage-free XAI machine learning framework for early software defect prediction by utilizing four NASA benchmark projects, which were cleaned to eliminate leakage.

The best performing algorithm in terms of balanced cross-project performance was XGBoost, showing the highest macro accuracy, precision, and F1-score. Random Forest had the highest ROC-AUC, LightGBM had the highest PR-AUC, and SVM had the highest recall. This

proves that model selection cannot be based on one measure but depends on the organization's defect cost profile. Statistically significant model difference was shown only for ROC-AUC, not for other measures due to limited number of projects.

The results of the explain ability analysis provided insights on how global and local SHAP outputs may reveal the software metrics which were used by the models for predictions. In the context of quality assurance workflow, the framework can identify priority modules for code review and give engineers explanations behind each critical score. Further research should include evaluation of temporal and cross-project generalization, probability calibration, threshold selection based on costs, explanation stability, and new code metrics.

Appendix A. Model and Analysis Settings

Table A1. Fixed model hyperparameters used in the reported analysis.

Component	Setting	Value
Cross-validation	Splits	Stratified 5-fold, shuffled; random seed 2026
Feature filter	Correlation threshold	Absolute Spearman rho > 0.90
Scaling	Transformer	RobustScaler: median and interquartile range
Class balancing	SMOTE	k_neighbors = 3; random seed 2026
Random Forest	Core settings	200 trees; max_features = sqrt; min_samples_leaf = 2
SVM	Core settings	RBF kernel; C = 10; gamma = scale
XGBoost	Core settings	250 trees; depth 4; learning rate 0.045; subsample/column sample 0.85; min_child_weight 2
LightGBM	Core settings	250 trees; 15 leaves; learning rate 0.045; subsample/column sample 0.85; min_child_samples 20
SHAP	Explanation sample	TreeSHAP on 600 held-out JM1 modules from a stratified 80/20 split

REFERENCES

- [1] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2-13, 2007. doi: 10.1109/TSE.2007.256941.
- [2] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, 2008. doi: 10.1109/TSE.2008.35.

- [3] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the NASA software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208-1215, 2013. doi: 10.1109/TSE.2013.11.
- [4] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, 1976. doi: 10.1109/TSE.1976.233837.
- [5] M. H. Halstead, *Elements of Software Science*. New York, NY, USA: Elsevier, 1977.
- [6] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200-1219, 2020. doi: 10.1109/TSE.2018.2876537.
- [7] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "The impact of correlated metrics on defect models," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 320-331, 2021. doi: 10.1109/TSE.2019.2891758.
- [8] J. Jiarpakdee, C. Tantithamthavorn, and A. E. Hassan, "AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models," in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, 2018, pp. 92-103. doi: 10.1109/ICSME.2018.00018.
- [9] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5-32, 2001. doi: 10.1023/A:1010933404324.
- [10] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273-297, 1995. doi: 10.1007/BF00994018.
- [11] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 785-794. doi: 10.1145/2939672.2939785.
- [12] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 3146-3154.
- [13] 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. doi: 10.1613/jair.953.
- [14] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 4765-4774.
- [15] S. M. Lundberg et al., "From local explanations to global understanding with explainable AI for trees," *Nature Machine Intelligence*, vol. 2, pp. 56-67, 2020. doi: 10.1038/s42256-019-0138-9.
- [16] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 166-185, 2022. doi: 10.1109/TSE.2020.2982385.
- [17] J. Shin, R. Aleithan, J. Nam, J. Wang, and S. Wang, "Explainable software defect prediction: Are we there yet?" *arXiv:2111.10901*, 2021.
- [18] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1-30, 2006.
- [19] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, no. 3, e0118432, 2015. doi: 10.1371/journal.pone.0118432.
- [20] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.
- [21] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering*, 2015, pp. 789-800. doi: 10.1109/ICSE.2015.91.

- [22] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 321-332. doi: 10.1145/2884781.2884857.

