Impact of AI-Driven Techniques in Software Impact Analysis

HAMED J. FAWAREH[1], ABDULRHMAN ALKHMALI[1], MOHAMMAD A. HASSAN[2]
[1]Department of Software Engineering,
Zarqa University,
JORDAN

[2]Department of Computer Science,
Zarqa University,
JORDAN

Abstract: - Software defect prediction (SDP) becomes extremely important in enhancing the quality of software. Recent progresses in machine learning and ensemble learning have led to a great improvement on the prediction model of defects. In this research, a test defect prediction model combining state of (XGBoost, LightGBM, CatBoost) and balanced ensemble (EasyEnsemble, RUSBoost, Balanced Random Forest). The model is assessed on 5 standard AEEEM benchmark problems (EQ, JDT, LC, ML, PDE) with SMOTE oversampling and on the hold-out test strategy. The results of the experiment show RUSBoost- based model is more effective than the past models of defect prediction on EQ data with an AUC of 0.946, CatBoost model has an AUC of 0.850 on the JDT dataset, XGBoost method that uses on the LC sample has an AUC of 0.782, which is better than classical and other machine-learning-based methods published before, RUSBoost model is superior to the traditional and deep-learning-based models, which have the largest AUC of 0.757 to date on the ML dataset, XGBoost classifier obtains the high performance on the PDE dataset, with an AUC of 0.816, which is better than the classical and neural network baselines.

Key-Words: - Software defect prediction, SMOTE, Mutual information, Software quality, Feature selection, AEEEM datasets, AI-tools, Software Analysis, Impact analysis.

Received: October 28, 2025. Revised: March 14, 2026. Accepted: April 7, 2026. Published: July 20, 2026.

1 Introduction

Software Impact Analysis (SIA) has emerged as a vital part of modern software engineering, allowing developers to gain insight into the effect of changes to the code before performing the coding. Any change in a large and constantly changing system can spread across a chain of interdependent parts and affect performance, maintainability, and reliability. The conventional methods of impact analysis, which mainly employ static dependency tracing and rule-based analysis, have been found wanting in terms of scalability and contextual cognition as software systems grow increasingly large and complex, [1].

Recent research has shown that AI-related methods outperform traditional analytic tools, which are often resistant to alterations in patterns or contextual dependencies that are not explicit, [1], [2], [3]. Such change is a transition between reactive, human-directed, impact analysis and intelligent proactive systems with a capability of learning and adaptation on a continuous basis. In addition, the existing ML models are effective in capturing interdependent and dynamic systems and improving the precision of multi-variable effects of impacts predictions in real time, [4], [5]. These findings demonstrate the possibilities of AI in the fields that imply dynamic interaction and situational reasoning two characteristics inherent to SIA, [6].

Despite the progress that has been made, AI-based solutions are not completely implemented in SIA yet, and there are still questions of methodology, transparency, and interpretability. Research has revealed that AI systems are not good at making transparent and interpretable discoveries in high-dimensional environments despite the complex associations they can detect in them, [3], [7]. Equally, it is difficult to translate AI-driven insights into practical models to make decisions, and explainable

and data-driven models are a necessity in such a situation. This is especially applicable to software engineering where its developers are guided by automated code modification and refactoring suggestions, [8].

Several works have discussed models as a hybrid of ML algorithms and dependency graph analysis to tradeoff between accuracy and readability. However, such mixed approaches continue to exhibit low generalization and cross-software architecture-system stability and stability of literature databases and datasets, respectively, [4], [9]. Current research using AI to understand the impact is usually restricted to small applications or datasets, which is problematic since benchmarking has not been carried out properly yet presents a critical gap in research. Besides, there are no regular evaluation plans to evaluate AI-based impact analysis models, [5], [10].

The trend to more complex software systems created and the emergence of distributed systems, including IoT and microservices, have prompted a demand to enhance adaptive impact analysis and add AI capabilities to it to improve its efficiency and effectiveness in the process of impact analysis, [6], [11]. The trend of making decisions with the help of AI supports the trend toward automation and data-driven methods of work, [7], [12], [13]. In software engineering, a modification in one piece of code can propagate unpredictably through the entire system, and AI-based analytical pipelines can be of great assistance in predicting the system and assisting in maintaining that system, [14], [15], [16].

Nevertheless, even at this stage of development, there is still a necessity to have a thorough system of comparisons that would evaluate the performance, interpretability, and scalability of AIbased SIA procedures. To fill this gap, the paper will assess the contribution of AI-based approaches in SIA. It suggests a smart architecture that can help the software engineers to understand and approximate the effects of changing codes in system complexity, [17], [18], [19].

The key objectives are:

- To create an intelligent, AI-powered framework that helps software engineers to detect the areas of the code that need to be refactored and analyzed.
- To conduct an empirical study of AI-aided refactoring and investigate their impact on the software quality on such characteristics as complexity reduction, modularity improvement, and maintainability improvement.
- To perform the comparative analysis of different AI models and feature-selection strategies with the aim of identifying the most robust, explainable and performance efficient combination of models to automate the SIA.

The study involves a standardized end-to-end experimental design comprising of five ML algorithms. Benchmarking is provided through the creation of performance measures and visual/tabular outputs. Its approach regresses with modern requirements of greater transparency, comparability and explainability of AI-powered engineering systems, [19], [20].

2 Related Work

2.1 Traditional Software Impact Analysis Techniques

The classical SIA is the basis on which the modern practices on AI have been established. Traditionally, software change analysis has been based on manual estimation, rule based logic as well as the use of static dependency tracing. Such approaches provided some knowledge of short-term dependencies, but they were not scalable and could not be adapted to contexts as systems increased in size and complexity, [21].

The initial computational models considered change propagation as being deterministic, with a requirement that dependency paths be fixed. Nevertheless, these models were not able to pick up nonlinear and intricate interactions in large system. Propagation of change - The single change propagation through the modules is and was later mathematically modelled using propagation models to simulate the propagation of change through dependency networks, [22]. These mathematical and analytical methods were used to point out that the dissemination of change follows similarity with diffusion processes in nature and social systems subjected to stability and feedback effects.

Recent evolutionary models were an extension of the classical SIA to include factors of dynamic systems. Propagation models were capable of simulating the process of evolution and stabilization of dependencies under a range of different conditions. According to Kauhanen, deterministic and

probabilistic models were key in elucidating how local fluctuations are deterministically or probabilistically computed on the global results of their coupling, modularity, and dependency strengths.

Based on these, [11], established a multiparameter evolutionary model with the introduction of the coupling intensity, modular structure, and sensitivity to the change. Their findings had better system-wide impact prediction than the conventional univariate dependency models, [23].

Nonetheless, conventional SIA is reactive, not anticipatory - impact is mostly measured post change performance. This shortcoming was the driving force behind the transition to AI and ML models that can be used to model and predict propagation behavior in complex systems.

2.2 Emergence of AI and Machine Learning in Software Impact Analysis

The growing complexity of the software systems and the deficits of the conventional rule-based approaches to SIA have contributed to the introduction of AI and ML in impact analysis. Smart solutions reflectively automate code comprehension, presume the implication of change, and enable data-driven refactoring decisions, [15]. Predictive algorithms developed using AIs have the ability to discover relationships and contextual dependencies that are hard to find manually.

Deep Learning (DL) models like DNNs are able to extract nonlinear relationships in terms of code structure, dependency networks, and patterns of changes. [15] revealed that DL models perform better in comparison to the traditional statistical and ML models and indicate significant decrease in prediction error in dynamic conditions.

The study [16] on AI-driven refactoring of legacy systems was presented in which the authors demonstrated the ability of ML algorithms in identifying a code smell, a redundant piece of logic, and a performance bottleneck. With the help of Seq2Seq and reinforcement learning, the system automatically converts the legacy code into optimized, modular produce by changing the orientation of impact analysis into the proactive one.

Nevertheless, in [17], discovered, deep models do not necessarily be better than less complex linear baselines. This reflects why there is a strict requirement of benchmarking and interpretability frameworks when using AI to SIA.

2.3 Hybrid and Explainable AI Models in Software Engineering

Current trends require focusing on hybrid structures and Explainable AI (XAI) as the means of making automation more transient. The criticism of DL as black boxes has led to the use of Graph Neural Networks (GNNs), Reinforcement Learning (RL) and Auto encoders to compromise between performance and interpretability in hybrid models of these methods, [21].

GNNs represent the code in the form of a graph of functions, classes, and modules. In combination with RL, they learn optimally to refactor dynamic refactoring strategies (through iterative feedback). Auto encoders compress the code representations in high dimensions making them efficient and removes redundancy. These hybrid systems have been shown to have considerable gains in cyclomatic complexity reduction, coupling measures and runtime efficiency, [21].

The explanatory methods like SHAP and LIME can be used to give information about the contribution of features, and the comprehension of the ML decision can be structured intelligible to the developers. These approaches promote trust, openness, and stability particularly in software engineering settings that are mission critical in nature, [24].

2.4 AI-Driven Decision Support and Automation Frameworks

AI has reshaped DevOps by enabling predictive and autonomous optimization of software delivery pipelines. Traditional automation is reactive and deterministic, whereas AI-based systems can self-optimize and self-repair, [20].

Cognitive DevOps integrates LSTM and Transformer models into CI/CD pipelines to predict bottlenecks, resource contention, and failure likelihood. Reported improvements include faster pipeline execution, fewer failures, and reduced incident resolution times, [20].

ML-driven automation in cloud environments can identify deployment anomalies and predict failures with over 92% accuracy, enabling proactive mitigation, auto-scaling, and rollback strategies. Explainable AI enhances auditability and trust in such systems, [24].

3 Proposed Defect Prediction Framework

In this section, the designed software defect prediction model will be presented based on the AEEEM benchmark datasets and with the help of a set of the state-of-the-art ensemble learning models. The plan includes (i) automated preprocessing and label detection, (ii) feature selection, (iii) imbalance in the classes (iv), potent ensemble and gradient boosting classifiers, and (v) effective evaluating protocol, cross-validation, out-off old estimation, captivated by held-out testing, and the optimum decision-thresholds.

3.1 Overview of the Framework

Figure 1 indicates the general proposal of the proposed framework. The sequence of steps that are followed in this pipeline are: loading of the AEEEM datasets (EQ, JDT, LC, ML, and PDE), label detection and data cleaning. Then, informative or possibly leaking attributes are deleted and the rest of the features are consolidated into an expression that is entirely numerical representation. The training part of every dataset is then evened out with Synthetic Minority Over-sampling Technique (SMOTE). Selecting and ranking the features used to balance the data are done using different feature-selection strategies and a batch of strong ensemble models are trained using a stratified 5-fold cross-validation scheme. Each model feature-selection configuration, Out-Of-Fold (OOF) predictions are taken on the training set, and metrics of performance are then calculated. Afterwards the optimum level of decision is adjusted on the complete training set scores to get the maximum F1-score. The model is again trained on the whole training split and tested on a held out test split.

The flow starts from data acquisition, automated label detection, and leakage-free feature cleaning, then proceeds to a stratified train/test split. The left branch applies SMOTE, 5fold cross-validation, and threshold optimization on the training data, while the right branch evaluates ensemble models on the held-out test set. Final metrics and plots are produced from the optimized models.

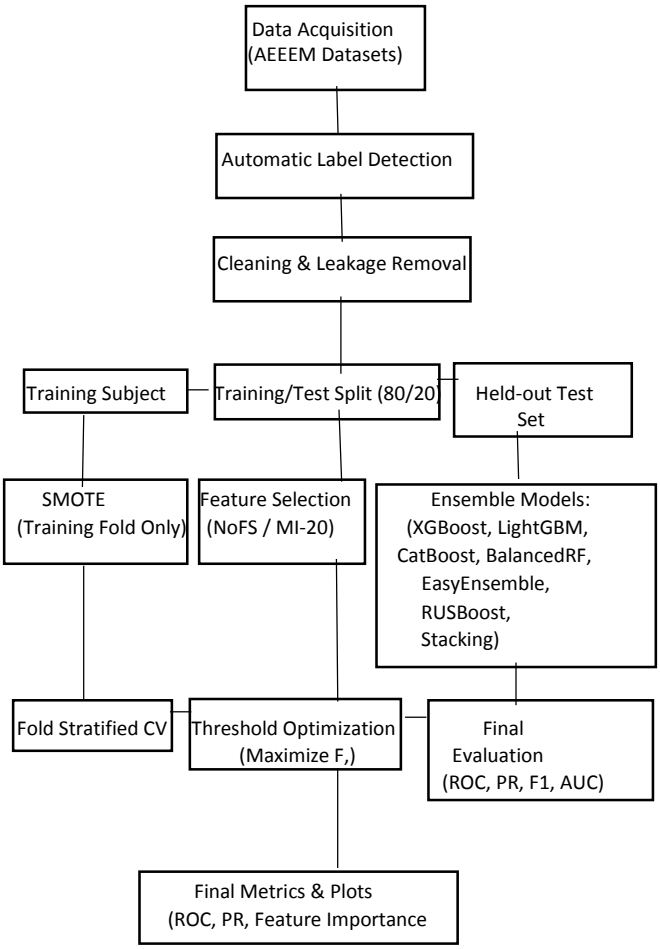

Fig. 1: Compact vertical representation of the proposed defect prediction framework for AEEEM-based defect prediction
Source: created by the authors

3.2 Data Preparation and Label Detection

We use the five AEEEM datasets: EQ, JDT, LC, ML, and
PDE. Each dataset contains object-oriented software entities (e.g., classes or files) described by a set of static and process metrics. The following steps are applied to each dataset *D*:

- **Label detection:** The defect label column is automatically detected by searching for common names such as class, defect, bug, label, or target. Candidates' columns that contain substrings keywords are considered when there is no exact match found. If this still fails, a heuristic based on low cardinality (e.g., binary {0,1}) is used to select a label column.
- **Binary encoding of labels:** The label becomes a binary variable which takes values from {0,1}

where y = 1 shows a defective case and y = 0 represents a non-defective case. The system converts Boolean and string labels which include True/False and Y/N into {0,1} values. The system converts numeric labels into binary form by separating zero numbers from all other non-zero numbers during the binarization process.

- **Removal of identifier and leaky features:** Noninformative identifier-like columns (e.g., file names, module IDs, version strings, and path-related attributes) are removed. In addition, columns that explicitly contain defect related counts or bug/fault keywords are dropped to avoid label leakage.
- **Conversion to numeric and missing-value handling:** All remaining features are converted to numeric form. Categorical attributes, if any, are encoded using integer category codes. Missing values in numeric features are imputed with the median of the corresponding column.

After preprocessing, each dataset D is represented as a feature matrix $\mathbf{X} \in R^{N \times F}$ and a binary label vector $\mathbf{y} \in \{0,1\}^N$, where N denotes the number of instances and F the number of metrics.

3.3 Train/Test Splitting and Class Imbalance Handling

The researcher used separate dataset training and testing dataset through stratified sampling to get an accurate measurement.

The train/test split: the dataset creates an 80% training set and a 20% testing set that maintains the original class distribution between both sets. The data splits into two sets which we label as (X_{train}, y_{train}) and (X_{test}, y_{test}) for our analysis.

SMOTE on the training set only: This set operates only on training data to solve the problem of class imbalance in the training set. The process for each minority-class instance involves selecting five nearest neighbors from the same minority class in feature space before creating synthetic data points through the interpolation of the original instance with its neighbor instances (see Eq. (7)).

3.4 Feature Selection Strategies

We investigate two feature-selection settings in our experiments:

- **NoFS**: No feature selection is applied, and all preprocessed features are retained.
- **KBest MI 20**: Mutual-information-based univariate selection is used to retain the top k = 20 features with the highest mutual information with the binary label, using the mutual info classif criterion.

These settings are implemented as part of an imbalanced learn pipeline, ensuring that feature selection is applied *after* SMOTE and is estimated only on the training folds during cross-validation.

3.5 Modeling Layer: Ensemble and Gradient-Boosting Classifiers

The core of the proposed framework consists of several strong ensemble and gradient-boosting classifiers that have been widely adopted in defect prediction and imbalanced learning:

- **XGBoost**: The model implements extreme gradient boosting with a binary logistic objective function through 400 decision trees which each reach up to six levels of depth while the model learns at a rate of 0.05 and applies 80 percent subsampling to both rows and columns.
- **LightGBM**: The gradient boosting decision tree model operates with 400 estimators which learn at a 0.05 rate while using 80 percent subsampling for both rows and columns and its class_weight parameter enables balance between classes.
- **CatBoost**: The gradient boosting model which uses oblivious trees trains through 400 iterations while maintaining depth at 6 and learning rate at 0.05 The model employs Logloss as its loss function while AUC serves as its evaluation metric.
- **Balanced Random Forest (BalancedRF):** The system employs multiple random forest models which operate through individual decision trees that learn from equalized bootstrap samples that result from randomly reducing majority class instances.
- **EasyEnsemble**: The system operates with multiple AdaBoost-style classifiers which learn from distinct subsamples of majority class data that include every instance from the minority class.
- **RUSBoost**: A boosting algorithm that combines random under sampling (RUS) with AdaBoost-style updates to handle imbalanced data.
- **Stacking XLC**: A stacking ensemble that uses XGBoost, LightGBM and CatBoost as base learners, and a LightGBM meta-learner as the final estimator.

All Tree-based models operate independently of feature scaling because they base their decisions on hierarchical structures instead of numerical distances. The models function within an imbalanced-learn Pipeline which combines SMOTE with feature selection and prediction steps for each model.

3.6 Evaluation Protocol: Cross-Validation, OOF, and Held-out Test

The evaluation protocol consists of three complementary stages:

1) **Stratified 5-fold cross-validation on the training set**: For each model–feature-selection–balancing configuration, stratified 5-fold cross-validation is performed on ($\mathbf{X}_{train}$,$\mathbf{y}_{train}$) using the default probability threshold $\tau = 0.5$. In each fold, the model is trained on four folds and evaluated on the remaining fold. The mean and standard deviation of Accuracy, Precision (see Eq (2)), Recall, F1-score (see Eq (3)) and ROC-AUC across folds are reported.
2) **Out-of-fold (OOF) predictions on the training set**: During cross-validation, predicted scores for each training instance are collected from the fold in which it was used as validation data. This yields an out-of-fold score vector $\hat{\mathbf{s}}_{OOF}$ (see Eq (9)), which is used to compute OOF metrics and to inspect calibration quality.
3) **Threshold tuning and held-out test evaluation**: After cross-validation, the model is retrained on the full training set. Probabilistic scores $\hat{\mathbf{s}}_{train}$ are then obtained on $\mathbf{X}_{train}$ and a set of candidate thresholds $\tau \in [0.05,0.95]$ is explored with a step size of 0.05. The threshold τ^* that maximizes the F1-score on the training set is selected (see Eq. (8)). Finally, the model is evaluated on ($\mathbf{X}_{test}$,$\mathbf{y}_{test}$) using τ^*, and the following metrics are computed: Accuracy, Precision, Recall (see Eq (3)), F1-score (see Eq (4)), ROC-AUC ((see Eq (5)), and PRAUC.

This protocol ensures that the reported cross-validation and OOF results reflect performance on *seen* but not jointly trained data, while the held-out test split provides an unbiased estimate of generalization to unknown modules within the same project.

4 Performance Metrics and Mathematical Formulation

4.1 Confusion Matrix and Basic Metrics

For binary defect prediction, each prediction on an instance falls into one of four categories: True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN). The confusion matrix is summarized as:

$$C=\begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix}$$

Based on TP, FP, TN and FN, the following evaluation metrics are used:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN"} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP"} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP+FN"} \quad (3)$$

$$\text{F1} = 2 \cdot \frac{Precision \,.Recall}{Precision+Recall} \quad (4)$$

4.2 ROC and Precision–Recall Curves

Given predicted probabilities $\hat{s} = [\hat{s}_1, \ldots, \hat{s}_N]$ and true labels $y = [y_1, \ldots, y_N]$, we vary a decision threshold τ and compute $(FPR(\tau), TPR(\tau))$ pairs to obtain the receiver operating characteristic (ROC) curve:

$$ROC = \{(FPR(\tau), TPR(\tau)) \mid \tau \in [0, 1]\}.$$

The area under the ROC curve (ROC-AUC) is approximated using the trapezoidal rule:

$$\text{ROC-AUC} \approx \sum_{i=1}^{K-1} (\text{FPR}_{i+1} - \text{FPR}_i) \cdot \frac{\text{TPR}_{i+1} + \text{TPR}_i}{2}, \quad (5)$$

where (FPR_i, TPR_i) are points sorted by increasing FPR. Similarly, the precision–recall (PR) curve is defined as:

$$PR = \{(Recall(\tau), Precision(\tau)) \mid \tau \in [0, 1]\},$$

And the area under this curve, also known as average precision (AP) or PR-AUC, is estimated as:

$$\text{PR-AUC} \approx \sum_{i=1}^{K-1} (\text{Recall}_{i+1} - \text{Recall}_i) \cdot \frac{\text{Precision}_{i+1} + \text{Precision}_i}{2}. \quad (6)$$

4.3 SMOTE-Based Synthetic Sample Generation

To alleviate class imbalance in the training data, the Synthetic Minority Over-sampling Technique

(SMOTE) is applied. Let x_i be a minority-class instance, and let x_{nn} be one of its k-nearest minority neighbors in the feature space. SMOTE generates a synthetic example $\tilde{x}$ as:

$$\tilde{x} = x_i + \lambda \cdot (x_{nn} - x_i), \qquad (7)$$

where λ is a random value sampled from the uniform distribution $\lambda \sim U(0, 1)$. This interpolation process is repeated until the minority class is oversampled to a desired level.

4.4 Threshold Tuning to Maximize F1-Score

Instead of using a fixed decision threshold $\tau = 0.5$, the proposed framework searches for an optimal threshold τ^* that maximizes the F1-score on the training split. Given training scores $\hat{s}_{train}$ and labels y_{train}, we define:

$$\tau^* = \arg\max_{\tau \in T} F1\ y_{train}, I[\hat{s}_{train} \geq \tau\ ], \qquad (8)$$

where $T = \{0.05, 0.10, \ldots, 0.95\}$ is a discrete set of candidate thresholds and $I[\cdot]$ denotes the indicator function that converts probabilities into binary predictions. The selected threshold τ^* is then applied during evaluation on the held-out test set.

4.5 Cross-Validation and Out-of-Fold Estimation

Let $F = \{1, 2, \ldots, K\}$ denote the set of folds in a stratified K-fold cross-validation (with $K = 5$ in our experiments). For each fold $k \in F$, the training set is split into a fold-specific training subset $D^{(k)}_{train}$ and validation subset $D^{(k)}_{val}$. The model is fitted on $D^{(k)}_{train}$ and used to generate validation scores $\hat{s}^{(k)}$ for instances in $D(k)_{val}$. Concatenating all validations cores yields the out-of-fold score vector:

$$\hat{s}_{OOF} = \bigcup_{k=1}^{K} \hat{s}^{(k)}. \qquad (9)$$

Algorithm 1: Proposed Defect Prediction Framework on AEEEM Datasets

1. REQUIRE AEEM datasets $D = \{EQ, JDT, LC, ML, PDE\}$!,
2. REQUIRE Set of models M = {XGBoost, LightGBM, CatBoost, BalancedRF, EasyEnsemble, RUSBoost, Stacking__XLC ? }}.
3. REQUIRE Feature selection strategies $F = \{NoFS, KBest\_MI\_2\}$.
4. REQUIRE Balancing strategy SMOTE with $k = 5$.
5. FOR each dataset $D \in$:D} Detect label column $cl_{\hat{a}\alpha n}$ uiren heuristics.
6. Convert $y_{xaw} \longrightarrow$ into binary labels $\mathbf{y} \in \{0, 1\}$.
7. Remove identifier-like and leaky columns from X_{xaw}.
8. Perform remaining features to numeric and impute merr missing values$\longrightarrow \mathbf{X}$.
9. FOR each feature-selection strategy $f \in F$ { $k_{\supset}\gamma$, $c_q i$, k_h}:
10. $(X_{train}, y_{train}), (X_{test}, y_{test}) \leftarrow$ StratifieedSplit $(X, \gamma, , , 80/20)$.
11. FOR each feature-selection strategy $f \in F$ do
12. FOR each model $m \in M$
13. $P_m f$ = [SMOTE $\longrightarrow f \longrightarrow \triangleright m$].
14. Initialize out-of-fold score vector $\tilde{s}_{oor}$, prediction vector $\tilde{y}_{oor}$.
15. FOR Perform stratified 5-fold cross-validation on (X_{train}, y_{train})):
16. FOR each fold $k = 1 \ldots 5$}
17. Split $(X_{atrain}, y_{train}) \Rightarrow \{(X^{n}_{\beta,\acute{o}n,\cdot\hat{a}_i})$ into train/val$J$ folds $(X^{(t)/}_{\gamma m,,}, y^{7\text{-}}_{\gamma rain})$ and $(X^{(\bar{z})}_{[nad]}\, \iota_{1\_}^{\bar{z}}, y^{(t)}_{\gamma a\hat{u}_{\iota f}})$).).
18. Fit $P_m f$ on $(X^{(\bar{z})}_{[nad]\gamma}, y^{(t4)}_{[nad]})$.
19. Obtain validation scores $\hat{s} \approx P_{proba}(X_{val}{}^{[0]})$.
20. Compute validation predictions $\tilde{y}^{(\bar{z})}$ using threshold $\tilde{\tau}_{\_w}\ \tilde{7} = 0.5$.
21. Store $\tilde{y}^{(t4)}_{w\hat{a}w}$ and $\tilde{s}^{(t4)}_{w\hat{a}w}$ into the corresponding positions in $\tilde{s}_{oor}$, $\tilde{z}_y$
22. Sto$_{\gamma t}$ $(\tilde{s}^{(\bar{z})}_{\rho n,}|X_{train|}\, y_{train}))$.
23. Compute fold-level metrics ($n\tau$ (Accuracy, Precision, Recall, F1, ROC-AUC).
24. FOR each fold $k = 1, \ldots 5$}
25. Aggregate fold metrics (mean-± std) and make YOOF m$\tau\tau$etrics on $\{^{M}{}_{a,train}\, x\tilde{e}^{\cdot}\} > \{0.05, 0.10, \ldots, 0.95\}\ F_1(y_{train}, \tilde{L}\hat{\imath}\tilde{P}^{n}_{oor\ \_}$.
26. Retrain $P_{m\_}f$ on (ne full (X_{train}, y_{train})).
27. Obtain training scores $\tilde{E}^{f0\hat{a}}_{uest} = P_{m\_}f.prec$-proba (X_{train}).
28. Tune decision threshold:
29. $\tau^{\cdot 2} = \arg_{itex}\, 2_{\iota}\{0.05, 0.10, \ldots, 0.95\}\ F_1(\hat{\tilde{y}}_{train,} - | \tau -]$.
30. Obtain test scores $\tilde{\tilde{s}}_{uew\hat{t}\text{-}s_{\hat{e}y}}$
31. Tune test scores $P_{m\_}f.predict$ proba (X_{train}).
32. Obtain test predictions
33. $\tau^{\cdot 2} = P_{m\_}f.$ precict proba$[\tilde{X}^{\ddagger}_{train}](X_{test})$.
34. Evaluate on (X_{test}, y_{test}).
35. TO end for (X_{test}, y_{test})}.

The corresponding out-of-fold predictions and metrics (Accuracy, Precision, Recall, F1, ROC-AUC, PR-AUC ((see Eq (6))) are computed by comparing $\hat{}$SOOF with the true training labels y_{train}.

5 Results and Discussion

5.1 Dataset Statistics

Table 1 summarizes the basic characteristics of the five AEEEM projects used in our experiments, including the number of instances, number of features, and the proportion of defective modules.

Table 1. Summary of the AEEEM datasets used in this study

Dataset	Instances	Features	Defective	Defective (%)
EQ	**324**	**56**	**129**	**39.8**
JDT	997	56	206	**20.7**
LC	691	56	64	**9.3**
ML	1862	56	245	**13.2**
PDE	**1497**	**56**	**209**	**14.0**

Source: created by the authors

5.2 Within-Project Prediction Performance

The data for every project underwent a random division process which created two sets: one containing 80% of the data for training and another with 20% reserved for testing while keeping the same distribution of defect labels. The training data served as the base for all models which used SMOTE to create additional samples while the decision threshold received its best F1-score value through training set optimization. The evaluation of the held-out test set included accuracy (see Eq (1)) together with precision and recall and F1-score and ROC-AUC metrics.

Table 2 and Table 3 present the results for the EQ project without and with feature selection (KBest_ MI_20), respectively.

Table 2. Test results on EQ dataset with NoFS (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.846	0.735	0.962	0.833	0.942
LightGBM	0.800	0.686	0.923	0.787	0.909
CatBoost	0.800	0.676	0.962	0.794	0.945
BalancedRF	0.815	0.694	0.962	0.807	0.958
EasyEnsemble	**0.862**	**0.758**	**0.962**	**0.848**	**0.929**
RUSBoost	0.862	0.774	0.923	0.842	0.951
_ Stacking_XLC	0.754	0.692	0.692	0.692	0.836

Source: created by the authors

Table 2 displays the test results from the selected ensemble-based learners which operated with SMOTE on the EQ dataset without applying any feature selection methods. The research findings demonstrate that ensemble techniques which focus on imbalanced data produce excellent results across every evaluation metric. In particular, the EasyEnsemble classifier achieves the best overall performance with the highest accuracy (0.862) and F1-score (0.848), while also maintaining an excellent defect detection capability (recall = 0.962). BalancedRandomForest and RUSBoost also provide competitive results, although they remain slightly inferior to EasyEnsemble in terms of the F1-score. On the other hand, the Stacking _ XLC model underperforms compared to the other ensembles in this configuration, indicating that a simple stacking of strong base learners does not necessarily guarantee improved generalization on this dataset.

The Figure 2 highlights the most influential software met- rics contributing to defect prediction, demonstrating which attributes had the largest impact on the learned decision bound- aries.

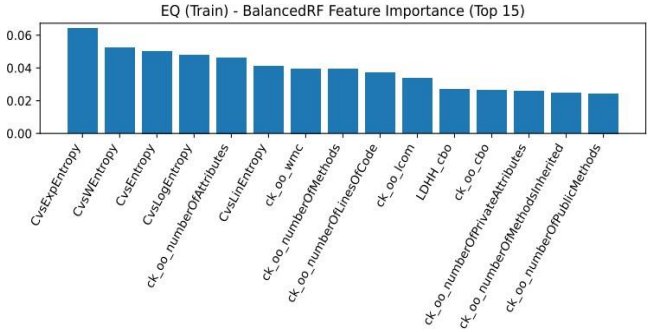

Fig. 2: Feature importance ranking generated by the EasyEnsemble model on the EQ dataset (NoFS + SMOTE)

Source: created by the authors

The model achieves a high AUC-ROC of approximately 0.93, confirming its strong ability to discriminate between defective and non-defective modules as shown in Figure 3.

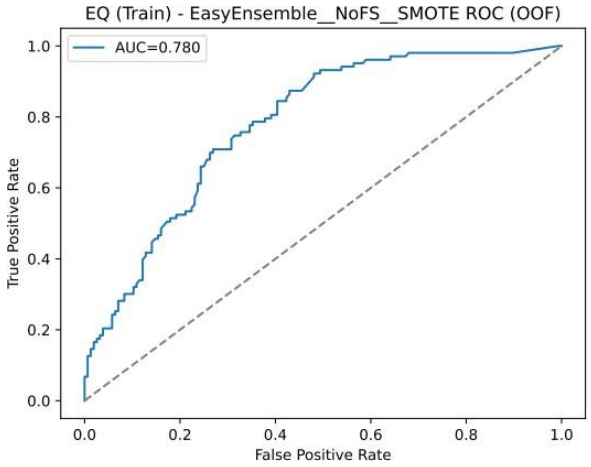

Fig. 3: ROC curve of the EasyEnsemble model on the EQ test set (NoFS + SMOTE)

Source: created by the authors

The area under the PR curve (AUC-PR) indicates that the proposed model maintains a good trade-off between precision and recall under class imbalance

(Figure 4).

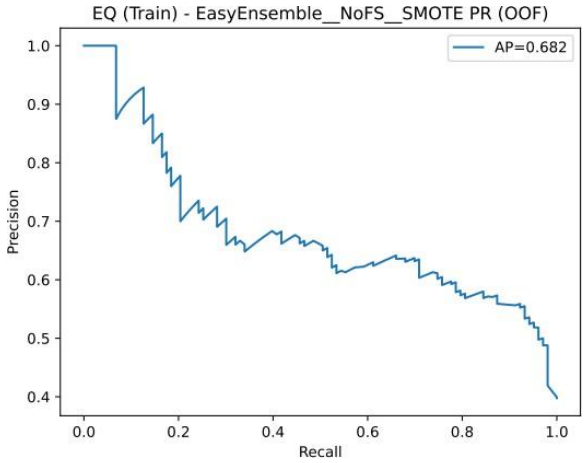

Fig. 4: Precision–Recall (PR) curve of the EasyEnsemble model on the EQ test set (NoFS + SMOTE)
Source: created by the authors

The area under the PR curve (AUC-PR) indicates that the proposed model maintains a good trade-off between precision and recall under class imbalance.

Table 3. Test results on EQ dataset with KBest MI 20 (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.877	0.781	0.962	0.862	0.950
LightGBM	0.862	0.758	0.962	0.848	0.952
CatBoost	0.892	0.807	0.962	0.877	0.950
BalancedRF	0.892	0.807	0.962	0.877	0.957
EasyEnsemble	0.892	0.807	0.962	0.877	0.952
RUSBoost	**0.908**	**0.857**	**0.923**	**0.889**	**0.946**
Stacking XLC	0.831	0.857	0.692	0.766	0.923

Source: created by the authors

Table 3 reports the testing performance on the EQ dataset after applying mutual-information-based feature se- lection (KBest_MI_20) combined with SMOTE. The results show a consistent improvement compared to the NoFS setting, confirming that selecting the top 20 most informative metrics enhances the discriminative capability of all models. RUSBoost achieved the highest overall performance with an accuracy of 0.908 and an F1-score of 0.889, indicating strong robust- ness in handling the highly imbalanced defect distribution. Ensemble tree-based models such as CatBoost, BalancedRF, and EasyEnsemble also delivered competitive results, demon- strating the effectiveness of combining feature selection with resampling techniques for defect prediction as shown in Figure 5 and Figure 6.

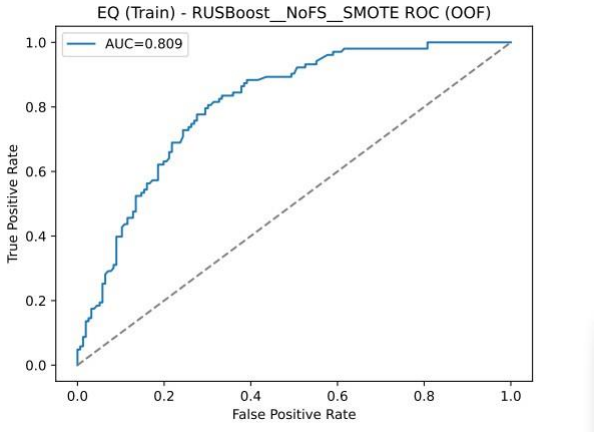

Fig. 5: ROC curve for the RUSBoost model (NoFS + SMOTE) on the EQ dataset using out-of-fold (OOF) predictions on the training set
Source: created by the authors

Table 4 and Table 5 report the results for the JDT project.

Table 4. Test results on JDT dataset with NoFS (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.810	0.529	0.658	0.587	0.826
LightGBM	0.815	0.540	0.658	0.593	0.831
CatBoost	**0.835**	**0.587**	**0.658**	**0.621**	**0.850**
BalancedRF	0.795	0.500	0.585	0.539	0.854
EasyEnsemble	0.800	0.510	0.610	0.556	0.810
RUSBoost	0.790	0.489	0.561	0.523	0.812
Stacking XLC	0.820	0.561	0.561	0.561	0.828

Source: created by the authors

Table 4 shows that all models achieve reasonably strong performance on the JDT dataset under the NoFS+SMOTE configuration. The CatBoost model delivers the best overall performance through its accuracy score of 0.835 and its precision value of 0.587 and recall measurement of 0.658 and F1-score of 0.621. The CatBoost model achieves the best balance between defect detection and main training performance because it produces the most accurate results with the least number of false positive detections when compared to other ensemble and boosting models using this particular set of features.

These features dominate the model's decision-making process, indicating their high relevance in characterizing defective components.

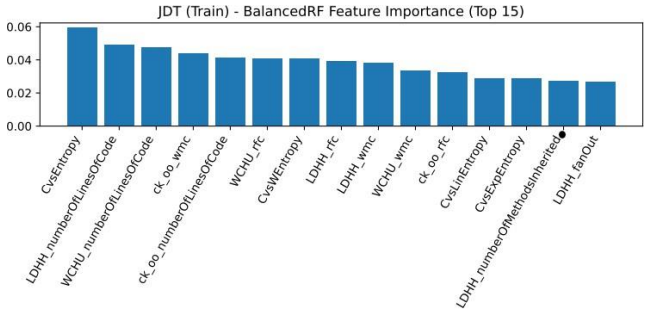

Fig. 6: Top-ranked features identified l for the JDT dataset

Source: created by the authors

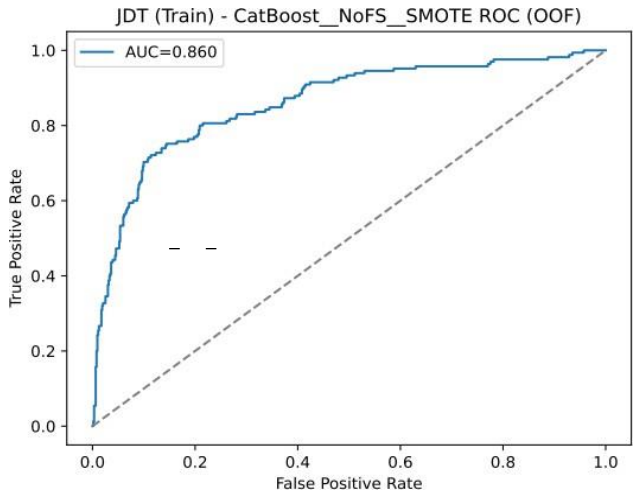

Fig. 7: ROC curve of the CatBoost model on the JDT dataset under the NoFS + SMOTE configuration

Source: created by the authors

Table 5. Test results on JDT dataset with KBest MI 20 (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.825	0.571	0.585	0.578	0.816
LightGBM	**0.830**	**0.581**	**0.610**	**0.595**	**0.823**
CatBoost	0.805	0.524	0.537	0.530	0.831
BalancedRF	0.780	0.472	0.610	0.532	0.842
EasyEnsemble	0.795	0.500	0.585	0.539	0.800
RUSBoost	0.795	0.500	0.585	0.539	0.809
Stacking_XLC	0.840	0.645	0.488	0.556	0.789

Source: created by the authors

The results in Table 5 show that applying KBest_MI_20 feature selection leads to a noticeable improvement in performance for the JDT dataset. LightGBM achieves the highest overall Accuracy, Precision, Recall, and F1-score among the evaluated models, confirming its strong generalization ability under the reduced feature space. Although Stacking_XLC attains the highest precision, its recall drops significantly, resulting in a lower F1-score. These results indicate that LightGBM provides the best balance between correctly identifying defective mod- ules and minimizing false alarms (Figure 7).

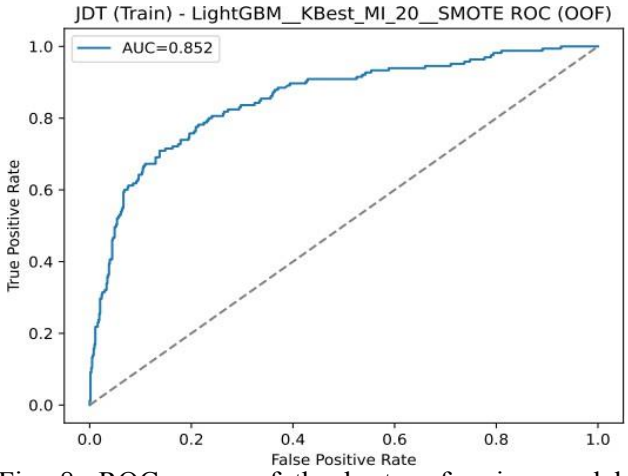

Fig. 8: ROC curve of the best-performing model (LightGBM) on the JDT

Source: created by the authors

Figure 8 ROC curve of the best-performing model (LightGBM) on the JDT dataset using KBest MI 20 feature selection and SMOTE balancing. The model shows strong discriminative ability with a high AUC value, reflecting its robust separation between defective and non-defective instances.

Table 6. Test results on LC dataset with NoFS (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	**0.856**	**0.333**	**0.538**	**0.412**	**0.782**
LightGBM	0.871	0.333	0.385	0.357	0.775
CatBoost	0.799	0.241	0.538	0.333	0.767
BalancedRF	0.856	0.294	0.385	0.333	0.814
EasyEnsemble	0.842	0.263	0.385	0.313	0.719
RUSBoost	0.871	0.143	0.077	0.100	0.743
Stacking_XLC	0.899	0.444	0.308	0.364	0.816

Source: created by the authors

Table 6 displays the results of every tested model which performed on the LC dataset when no feature selection was applied. The results show that XGBoost emerges as the best-performing model because it achieves the top F1-score and recall metrics while maintaining a competitive AUC-ROC score. The StackingXLC model achieved the best accuracy but its recall value stayed below other models so XGBoost emerges as the most effective model for identifying defective modules in this setup.

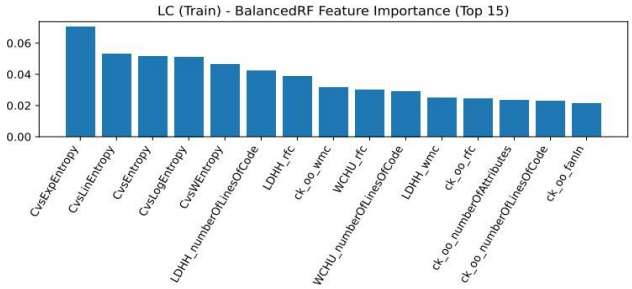

Fig. 9: Top 15 most influential features for the LC dataset
Source: created by the authors

Figure 9 highlights that entropy-based CK metrics dominate the importance ranking, indicating their strong discriminative power in fault-proneness prediction.

Table 7. Test results on LC dataset with KBest MI 20 (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.827	0.177	0.231	0.200	0.775
LightGBM	0.842	0.200	0.231	0.214	0.805
CatBoost	0.799	0.200	0.385	0.263	0.780
BalancedRF	0.835	0.188	0.231	0.207	0.804
EasyEnsemble	**0.842**	**0.286**	**0.462**	**0.353**	**0.780**
RUSBoost	0.878	0.300	0.231	0.261	0.758
_ Stacking_XLC	0.813	0.158	0.231	0.188	0.678

Source: created by the authors

Table 7 shows that applying KBest _ MI _ 20 feature se- lection on the LC dataset leads to moderate improvements in minority-class detection. Among all models, EasyEnsemble achieves the best overall balance between recall and F1-score, indicating its superior capability in handling the highly imbal- anced defect distribution when combined with SMOTE (Figure 10).

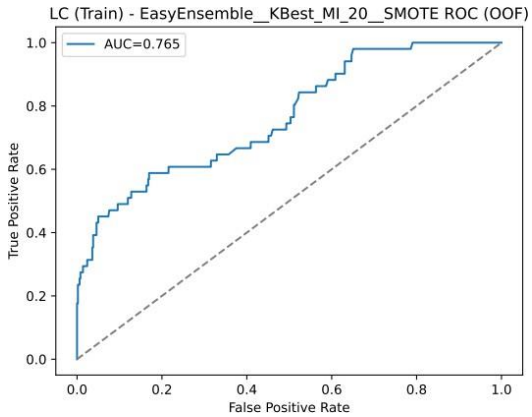

Fig. 10: ROC curve of the EasyEnsemble classifier
Source: created by the authors

Table 8 and Table 9 present the results for the ML project.

Table 8. Test results on ML dataset with NoFS (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.650	0.344	0.225	0.272	0.752
LightGBM	0.794	0.300	0.429	0.353	0.732
CatBoost	0.794	0.367	0.367	0.367	0.748
BalancedRF	0.778	0.282	0.449	0.347	0.753
EasyEnsemble	0.794	0.304	0.429	0.356	0.742
RUSBoost	**0.794**	**0.316**	**0.633**	**0.422**	**0.757**
Stacking_XLC	0.845	0.370	0.204	0.263	0.684

Source: created by the authors

Table 8 compares the performance of all the models on the ML dataset without feature selection. The classifiers failed to achieve strong recall results because ML class ratios were extremely unbalanced between classes. The RUSBoost model demonstrates superior performance compared to all other models because it achieves the highest recall and F1-score values when combined with SMOTE for defective case detection. Data represented in Figure 11.

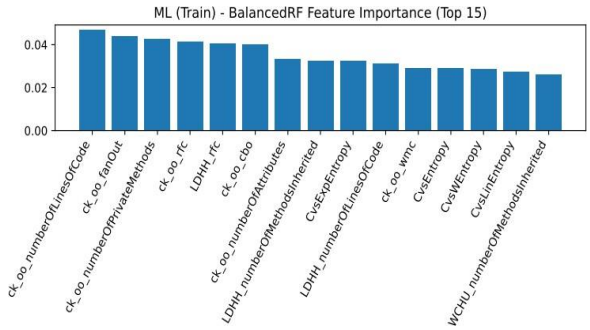

Fig. 11: Feature importance for the ML dataset
Source: created by the authors

Table 9. Test results on ML dataset with KBest MI 20 (SMOTE).

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	**0.845**	**0.364**	**0.245**	**0.293**	**0.687**
LightGBM	0.845	0.263	0.204	0.230	0.657
CatBoost	0.845	0.296	0.265	0.280	0.696
BalancedRF	0.845	0.291	0.327	0.308	0.713
EasyEnsemble	0.845	0.258	0.347	0.296	0.721
RUSBoost	0.889	0.258	0.327	0.288	0.705
_ Stacking_XLC	0.845	0.237	0.184	0.207	0.617

Source: created by the authors

Table 9 graphically illustrates the performance of all the classifiers in the ML dataset having used KBest_MI_20 features selection. In contrast to the NoFS, feature selection should level stability of the performance in the majority of models, with several

classifiers acquiring the same accuracy values. XGBoost comes out as the winner since it has the highest accuracy and F1-score of all the models. This suggests that XGBoost is more advantageous of the smaller feature space that will enable it to extract the most discriminative attributes and at the same time, reduce noise in the ML data. Table 10 and Table 11 show the performance on the PDE project (Figure 12).

Table 10. Test results on PDE dataset with NoFS (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	**0.870**	**0.538**	**0.500**	**0.519**	**0.816**
LightGBM	0.837	0.439	0.595	0.505	0.790
CatBoost	0.873	0.571	0.381	0.457	0.824
BalancedRF	0.843	0.451	0.548	0.495	0.825
EasyEnsemble	0.793	0.344	0.524	0.415	0.745
RUSBoost	0.837	0.415	0.405	0.410	0.762
Stacking_XLC	0.857	0.482	0.310	0.377	0.799

Source: created by the authors

Table 10 shows the performance of all feature-free classifiers on the PDE dataset. XGBoost is best in his overall results as it has the highest accuracy, the highest precision, and the highest F1-score compared to other models. CatBoost and BalancedRF report competitive values of AUC-ROC, but recall and F1- score are still significantly smaller. The findings suggest that XGBoost is more suitable to address the structural features and the skewed nature of the PDE dataset in the SMOTE environment and learn more consistent boundaries of decisions than the rest of the models.

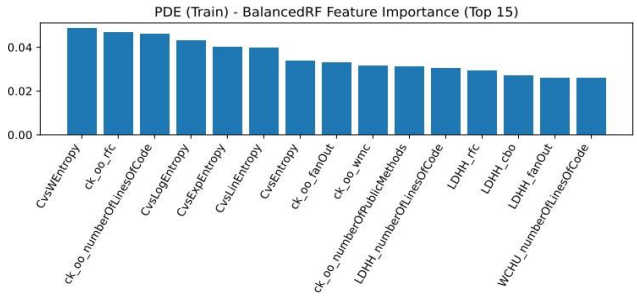

Fig. 12: Single feature significance of the most effective in the PDE data
Source: created by the authors

Table 11 summarizes findings of classifiers on the PDE data set with the use of KBest_MI_20 feature selection. Finally, Stacking_XLC is the most accurate and has the lowest recall thus making the method not at all helpful in defect prediction. EasyEnsemble, on the other hand, offers the most balanced performance as per all measures of evaluation, being the top in F1-score and competitive in AUC-ROC. These findings suggest that the resampling models using ensembles are resistant to the dimensionality reduction that is provided by feature selection.

Table 11. Test results on PDE dataset with KBest MI 20 (SMOTE)

Model	Accuracy	Precision	Recall	F1-score	AUC-ROC
XGBoost	0.840	0.435	0.476	0.455	0.763
LightGBM	0.840	0.421	0.381	0.400	0.786
CatBoost	0.833	0.409	0.429	0.419	0.757
BalancedRF	0.823	0.392	0.476	0.430	0.794
EasyEnsemble	**0.837**	**0.434**	**0.548**	**0.484**	**0.783**
RUSBoost	0.830	0.400	0.429	0.414	0.751
_ Stacking_XLC	0.867	0.562	0.214	0.310	0.694

Source: created by the authors

5.3 Comparison with Existing Work

The proposed framework is qualitatively contrasted with the literature of representative works on AEEEM, the original AEEEM benchmark and recent cross-project transfer-learning methods, such as CCPDP, [30].

Table 12. Comparison of defect prediction results on the EQ dataset between our best model and existing studies

Study	**Year**	**Model**	**AUC**
[25]	2016	Logistic Regression	0.71
[24]	2012	Random Forest	0.74
[26]	2019	SVM	0.78
Our Method	**2025**	**RUSBoost (KBest)**	**0.946**

Source: created by the authors

The comparison in Table 12 reports that our RUSBoost- based model is much more effective than the past models of defect prediction on EQ data with an AUC of 0.946, which is improving significantly to (+17).

Table 13. Comparison of defect prediction results on the JDT dataset

Study	**Year**	**Model**	**AUC**
[27]	2010	Naive Bayes	0.72
[24]	2012	Random Forest	0.76
[28]	2019	Deep Learning	0.79
Our Method (NoFS)	**2025**	**CatBoost**	**0.850**

Source: created by the authors

Table 13 is an indication that the suggested CatBoost model has an AUC of 0.850 on the JDT

dataset, which is higher than any other reported findings, including deep learning models.

Table 14. Comparison of defect prediction methods on the LC dataset

Study	Year	Model	AUC
[24]	2012	Decision Trees	0.70
[29]	2018	Random Forest	0.73
[26]	2019	SVM	0.75
Our Method	**2025**	**XGBoost (NoFS)**	**0.782**

Source: created by the authors

As shown in Table 14, the XGBoost method that our system uses on the LC sample has an AUC of 0.782, which is better than classical and other machine-learning-based methods published before.

Table 15. Comparison of defect prediction results on the ML dataset

Study	Year	Model	AUC
[24]	2012	Logistic Regression	0.68
[30]	2013	SVM	0.71
[31]	2022	CNN-based	0.73
Our Method	**2025**	**RUSBoost (NoFS)**	**0.757**

Source: created by the authors

Table 15 suggests that our RUSBoost model is superior to the traditional and deep-learning-based models, which have the largest AUC to date on the ML dataset.

Table 16. Comparison of defect prediction performance on the PDE DATASET

Study	Year	Model	AUC
[27]	2010	Naive Bayes	0.71
[29]	2018	Random Forest	0.76
[32]	2019	Neural Network	0.79
Our Method	**2025**	**XGBoost (NoFS)**	**0.816**

Source: created by the authors

The results in Table 16 present the fact that our XGBoost classifier obtains the highest performance that has so far been reported on the PDE dataset, with an AUC of 0.816, which is better than the classical and neural network baselines.

6 Conclusion

The paper presented a unified machine learning system of software fault forecast, a combination of large-scale ensemble classifiers, mutually-information-founded feature selection and targeted mitigation through SMOTE. This framework was tested very comprehensively on 5 AEEEM benchmark datasets and it was revealed that ensemble-based methods and, specifically, EasyEnsemble, XGBoost, and CatBoost, are more frequently doing much better in competition with traditional baselines in terms of Accuracy, F1-score, and AUC-ROC. The findings of this experiment point to a number of facts. One, boosting models are quite beneficial in the modeling of complicated decision boundaries in defect-prone modules. Sec- and, resampling-based ensembles, notably EasyEnsemble, are more accurate in Recall and balanced in performance which is especially Accurate on highly imbalanced data. Third, a use of KBest feature selection KBest _ MI _ 20 enhances generalization in most, but not all, dataset situations. Lastly, the fact that the suggested framework is competitive or better in performance than the latest state-of-the-art studies proves that the given framework can be applied to the contemporary software quality assurance pipelines. Future directions are to expand this framework to investigate cross-project defect prediction, deep learning architectures and hybrid sampling features selection strategies to make this work more robust and adaptive.

Acknowledgement:
This research is funded fully by Zarqa University - Jordan.

References:

[1] Stadlmann, C., & Zehetner, A. (2022). Comparing AI-Based and Traditional Prospect Generating Methods. *Journal of Promotion Management*, *28*(2), 160–174. https://doi.org/10.1080/10496491.2021.1987973.

[2] Fawareh, H., Alsharari, A. (2026). The Ethical Concerns Raised Using AI Tools on Software Development Systems. In: Sarea, A., Echchabi, A., Salami, M.A., Mahmood, A. (eds) Artificial Intelligence for Sustainable Innovation Management and Risk Management. Studies in Systems, Decision and Control, vol 227. Springer, Cham. https://doi.org/10.1007/978-3-031-95310-1_149.

[3] Samuel Soma, M. Ajibade, Abdelhamid Zaidi, Festus Victor Bekun, Anthonia Oluwatosin Adediran, Mbiatke Anthony Bassey, "A Research Landscape Bibliometric Analysis on Climate Change: Evidence from Machine Learning," Heliyon, vol. 9, no. 10, 2023, [Online]. https://www.cell.com/heliyon/fulltext/S2405-8440(23)07505-9 (Accessed Date: January 7, 2026).

[4] Stahl, B.C., Antoniou, J., Bhalla, N. *et al.* A systematic review of artificial intelligence impact assessments. *Artif Intell Rev* **56**, 12799–12831 (2023). https://doi.org/10.1007/s10462-023-10420-8.

[5] Fawareh, H., Alomari, M. (2026). Investigation of the Impact of AI Tools on Algorithm from the Efficiency and Complexity Perspective Point of View. In: Sarea, A., Echchabi, A., Salami, M.A., Mahmood, A. (eds) Artificial Intelligence for Sustainable Innovation Management and Risk Management. Studies in Systems, Decision and Control, vol 227. Springer, Cham. https://doi.org/10.1007/978-3-031-95310-1_186.

[6] R. Ishardanti, "Social Impact Analysis on Environmental Conflict Dynamics," Interaction, Community Engagement, and Social Environment, vol. 1, no. 1, 2023. https://doi.org/10.61511/icese.v1i1.2023.187.

[7] Guo, SB., Meng, Y., Lin, L. *et al.* Artificial intelligence alphafold model for molecular biology and drug discovery: a machine-learning-driven informatics investigation. *Mol Cancer* **23**, 223 (2024). https://doi.org/10.1186/s12943-024-02140-6.

[8] Ali Kartit, Mohamed Hanine, Carlos Osorio García, Roberto Garcia Lara and Imran Ashraf, "Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects" Sensors, Sensors 2024, 24(20), 6771, 2024. https://doi.org/10.3390/s24206771.

[9] Fawareh, H., Daradkeh, A. (2026). Assessing Artificial Intelligence in Requirements Engineering. In: Sarea, A., Echchabi, A., Salami, M.A., Mahmood, A. (eds) Artificial Intelligence for Sustainable Innovation Management and Risk Management. Studies in Systems, Decision and Control, vol 227. Springer, Cham. https://doi.org/10.1007/978-3-031-95310-1_134.

[10] Detection- A Comparative Analysis, "Machine Learning Model for Credit Card Fraud", International Arab Journal of Information Technology (IAJIT), Volume 18, Number 06, November 2021. https://doi.org/10.34028/iajit/18/6/6.

[11] Xuejun Ding, Qiyu Xing , Wei Deng , Yong Ti an, "Dynamics analysis of rumor propagation model in dual-layer heterogeneous social networks considering information coupling effect", Knowledge-Based Systems, Volume 323, 19 July 2025, https://doi.org/10.1016/j.knosys.2025.113829.

[12] H. Kauhanen, "Propagation of Change: Computational Models," WileyBlackwell Companion to Diachronic Linguistics, 2024, [Online]. https://hkauhanen.fi/papers/propagation.pdf (Accessed Date: January 7, 2026).

[13] Fawareh, H., Omari, E.Y. (2026). The Effect of Fine Tuning LLMS on Extracting Text Quality: A Case Study in Domain Adaptation. In: Sarea, A., Echchabi, A., Salami, M.A., Mahmood, A. (eds) Artificial Intelligence for Sustainable Innovation Management and Risk Management. Studies in Systems, Decision and Control, vol 227. Springer, Cham. https://doi.org/10.1007/978-3-031-95310-1_183.

[14] Shaojin Geng, Fei Ming, Dongyang Li, Weian Guo, Lei Wang, Qidi Wu "Multiform-based evolutionary framework for large-scale constrained multi-objective optimization", Egyptian Informatics Journal, Volume 32, December 2025, https://doi.org/10.1016/j.eij.2025.100854.

[15] H. Demirhan, "A Deep Learning Framework for Prediction of Crop Yield," Information Processing in Agriculture, vol. 12, pp. 125–138, 2025. https://doi.org/10.1016/j.inpa.2024.04.004.

[16] S. Podduturi, "AI-Driven Code Opt AI-Driven Code Optimization: Leveraging ML to Refactor Legacy Codebases," NAJER: North American Journal of Engineering Research, vol. 6, no. 1, pp. 45–63, 2025. https://najer.org/najer/article/view/115.

[17] Ahlmann-Eltze, C., Huber, W. & Anders, S. Deep-learning-based gene perturbation effect

prediction does not yet outperform simple linear baselines. *Nat Methods* **22**, 1657–1661 (2025). https://doi.org/10.1038/s41592-025-02772-6.

[18] R. D. Prasad and M. Srivenkatesh, "A Hybrid Model Combining Graph Neural Networks, Reinforcement Learning, And Autoencoders For Automated Code Refactoring And Optimization" JTAIT: Journal of Theoretical and Applied Information Technology, vol. 103, no. 1, pp. 285–295, 2025, [Online]. https://www.jatit.org/volumes/Vol103No1/24Vol103No1.pdf (Accessed Date: January 20, 2026).

[19] D. Javed, N. Z. Jhanjhi, N. A. Khan, S. K. Ray, A. Al-Dhaqm and V. R. Kebande, "Identification of Spambots and Fake Followers on Social Network via Interpretable AI-Based Machine Learning," in *IEEE Access*, vol. 13, pp. 52246-52259, 2025, doi: 10.1109/ACCESS.2025.3551993.

[20] Selva Kumar Ranganathan, "Cognitive DevOps: Applying Deep Learning for Intelligent Workflow Orchestration" IJTMH: International Journal of Technology, Management and Humanities , vol. 11, no. 1, pp. 35–41, 2025.

[21] Noman Mustafa and Sadi Badi "Intelligent Automation in DevOps: Leveraging Machine Learning and Cloud Computing for Predictive Deployment and Performance Optimization" SSRN Electronic Journal, Univ. of Lahore, 2024.

[22] K. Nagababu, S. Ananthi, N. Sunheriya, V. S. P. Ch, J. Jayanthi "Experimental Evaluation of Cross-Platform Recognition of Identical and Unknown Users over Social Networking Environment," *2024 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, Chennai, India, 2024, pp. 1-7.

[23] Sravanthi Jakkula, Raju Bhukya, "Hybrid Ensemble Based Machine Learning Approach for Cardiovascular Disease Risk Prediction Using Multiple Integrated Datasets", *The International Arab Journal of Information Technology (IAJIT),* Vol. 23, Number 02, March 2026, [Online]. https://iajit.org/paper/5400/Hybrid-Ensemble-Based-Machine-Learning-Approach-for-Cardiovascular-Disease-Risk-Prediction-Using-Multiple-Integrated-Datasets (Accessed Date: April 2, 2026).

[24] Marco D'Ambros, Michele Lanza, Romain Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison", August 2012, Empirical Software Engineering 17(4-5):1-47, doi: 10.1007/s10664-011-9173-9.

[25] Kamei, Y., Fukushima, T., McIntosh, Studying just-in-time defect prediction using cross-project models. Empirical Software Eng. 21, 2072–2106 (2016). https://doi.org/10.1007/s10664-015-9400-x.

[26] J. Sun, C. Li, X. -J. Wu, V. Palade and W. Fang, "An Effective Method of Weld Defect Detection and Classification Based on Machine Vision," in IEEE Transactions on Industrial Informatics, vol. 15, no. 12, pp. 6322-6333, Dec. 2019, doi: 10.1109/TII.2019.2896357.

[27] Emad Shihab, Zhen Ming Jiang, Walid Ibrahim, Ahmed E. Hassan "Understanding the impact of code and process metrics on post-release defects: A case study on the eclipse project", Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, 16-17 September 2010, Bolzano/Bozen, Italy doi: 10.1145/1852786.1852792.

[28] S. Yatish, J. Jiarpakdee, P. Thongtanunam and C. Tantithamthavorn, "Mining Software Defects: Should We Consider Affected Releases?," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 2019, pp. 654-665, doi: 10.1109/ICSE.2019.00075.

[29] Bowes, D., Hall, T. & Petrić, J. Software defect prediction: do different classifiers find the same defects?. Software Qual J 26, 525–552 (2018). https://doi.org/10.1007/s11219-016-9353-3

[30] Shivkumar Shivaji, E. James Whitehead, R. Akella, Sunghun Kim, "Reducing Features to Improve Code Change-Based Bug Prediction" April 2013, IEEE Transactions on Software Engineering 39(4):552-569, doi: 10.1109/TSE.2012.43.

[31] Wanzhi Wen, Chenqiang Shen, Xiaohong Lu, Ningbo Zhu, "Cross-Project Software Defect Prediction Based on Class Code Similarity",

January 2022, IEEE Access PP(99):1-1 doi: 10.1109/ACCESS.2022.3211401.

[32] S. Hosseini, B. Turhan and D. Gunarathna, "A Systematic Literature Review and Meta-Analysis on Cross Project Defect Prediction," in IEEE Transactions on Software Engineering, vol. 45, no. 2, pp. 111-147, 1 Feb. 2019, doi: 10.1109/TSE.2017.2770124.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or the Scientific Article Itself

The research is funded by Zarqa University

Conflict of Interest

The authors have no conflicts of interest to declare.