International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(10s) 2026 pp. 333-355
https://doi.org/10.70917/ijcisim-2026-3588
Article

Bayesian-Optimized Hybrid Random Forest and Feature Attention-Based Residual Neural Network for Software Defect Prediction

Dinesh Kumar[1], Saurabh Charaya[2]

[1]Department of Computer Science and Applications, Om Sterling Global University, Hisar, India, swiftdkumar@gmail.com
[2]School of Engineering and Technology, Om Sterling Global University, Hisar, India, Saurabh.charaya@gmail.com
***Corresponding Author:** Dinesh Kumar, E-Mail: swiftdkumar@gmail.com

Abstract: Software defect prediction is an important activity used for identifying fault-prone modules at an early stage of software development. It is used for enhancing software reliability and reducing maintenance costs. Traditional software defect prediction machine learning(ML) models perform well but often fail to capture complex nonlinear relationships in software metrics. Deep learning(DL) models address this limitation by learning hierarchical features but can still be hindered by class imbalance, poor hyperparameter tuning, and insufficient attention to feature importance. To address these challenges, this paper proposes a hybrid software defect prediction model (RF-FA-ResANN) that integrates Random Forest with a Feature Attention-based Residual Artificial Neural Network. This model is evaluated using Stratified 5-Fold Cross-Validation with Accuracy, Precision, Recall, F1-score, and ROC-AUC as performance metrics. Experimental results demonstrate that this model achieved accuracies of 99.83% and 90.29%, F1- scores of 99.15% and 87.1%, with ROC-AUC scores of 0.99998 and 0.94448 on software_defect and NASA JM1 datasets respectively. Experimental results show that proposed RF-FA-ResANN model outperformed conventional Machine Learning models (Support Vector Machine, Naïve Bayes, and Random Forest) as well as Deep Learning models (ANN, Attention-based ANN, Residual ANN, and Attention-based Residual ANN). This model integrates ensemble learning, feature attention, residual learning, and Bayesian optimization, which improves robustness and accuracy in software defect prediction.
Keywords: Software Reliability, Software Fault Prediction, Bayesian Optimization, ANN, Support Vector Machine, Random Forest, Residual ANN, SMOTE, Feature Attention, Stratified Cross-Validation.

1. Introduction

1.1 Background

Software defects, also referred to as software faults or bugs. Software defects have the potential to substantially affect software performance, security, maintainability, and its operational stability. In critical application domains such as healthcare, banking, aerospace, transportation, cloud computing, and defense systems, even a minor software defect may lead to fatal operational and financial consequences. Therefore, early identification of defect-prone software modules has become a crucial research objective in software engineering. Software fault prediction techniques assist developers and testers in identifying risky modules in early stages of SDLC. It minimizes debugging costs and testing effort. It also reduces maintenance overhead, and software failure risks.

Traditional software fault prediction techniques were mostly dependent on statistical models, expert systems, rule-based classification systems, and historical defect analysis methods. From simple statistical estimation techniques, software fault prediction has matured into a sophisticated prediction framework.

Early studies implemented fuzzy rule-based classification approaches and statistical estimation models for software defect classification [1]. A substantial transition in the field came with introduction of a technology called Machine Learning.

ML algorithms learn hidden patterns from software metrics and defect history automatically. They do not rely on manually defined rules. ML utilizes software complexity measures, code churn information, object-oriented metrics, and defect history. Experimental studies show that ML methods achieve higher prediction accuracy than traditional statistical methods [2].

Defect prediction has been significantly used with various ML algorithms like Decision Trees, Random Forest, Logistic Regression, ANNs, SVM, and Naive Bayes. Individual ML models often suffer from problems like overfitting, redundant features, sensitivity of parameters, and lack of generalization ability. To address these problems, hybrid ML frameworks with optimization algorithms have been suggested in recent years. Subsequently, Deep Learning (DL) techniques were added. DL models showed better performance and prediction accuracy.

1.2 Problem Statement

Several challenges make accurate software defect prediction difficult:

a) Software datasets are usually highly imbalanced, which results in a biased model learning [14].

b) Traditional ML algorithms cannot capture the complex nonlinear relationships between features.

c) There are drawbacks to deep neural networks including overfitting and vanishing gradient.

d) Mostly existing models mainly focus on ML or DL alone, which restricts the predictive ability of the models.

e) Manually tuning the hyperparameters is costly and ensures that you may not get the best model [16].

Hence, a hybrid architecture that successfully integrates the ensemble learning [23], attention mechanism [12], residual learning, and automatic hyperparameter tuning can enhance the accuracy of software defect prediction [16].

1.3 Research Gap

Although existing ML and DL based software defect prediction models have achieved encouraging results, several limitations remain. Traditional ML models cannot effectively capture complex nonlinear feature interactions [3]. DL models help overcome this limitation but their performance can still be affected by class imbalance, poor hyperparameter tuning, and insufficient attention to feature importance. Existing attention-based and residual neural networks improve feature learning and optimization. But these frameworks are rarely integrated with ensemble learning strategies. Moreover, few studies combine Random Forest probability-based feature augmentation, feature attention, residual learning, Bayesian optimization, and SMOTE within a single end-to-end framework. To bridge identified research gap, this work introduces a hybrid RF-FA-ResANN framework. This hybrid framework unifies ensemble learning, deep learning, and automated optimization techniques within a single framework.

1.4 Research Objectives

This research focuses on design, implementation, and evaluation of a novel software defect prediction framework. This framework integrates Random Forest, a Feature Attention-based Residual Artificial Neural Network (FA-ResANN) and a Bayesian Optimization into a single framework. It combines ensemble learning, feature attention, residual learning, automated hyperparameter optimization, and class imbalance handling. These techniques work together to improve accuracy, robustness, and generalization capability of software defect prediction. The proposed model and some conventional ML and DL approaches are evaluated using two benchmark Software defect datasets. Then these results are compared.

1.5 Contribution of Research

This research presents a hybrid software defect prediction framework that combines the strengths of ensemble learning and deep learning. The proposed Random Forest–Feature Attention–Residual Artificial Neural Network (RF-FA-ResANN) model employs Random Forest to generate probability-based feature information. This feature information is integrated with original software metrics before classification. A Feature Attention-based Residual ANN is then used to learn discriminative feature representations. Bayesian Optimization is used to automatically identify suitable network hyperparameters. In addition, SMOTE is applied to reduce impact of class imbalance. Stratified 5-Fold Cross-Validation is adopted to obtain reliable and unbiased performance estimates. The proposed model combines these complementary techniques into a single prediction model. As a result, it aims to deliver more accurate, reliable, and generalizable software defect prediction.

1.6 Paper Organization

Section 2 reviews related work and also discusses limitations of existing ML and DL approaches. **Section 3** describes proposed RF-FA-ResANN model. It also describes, Work flow, Dataset descriptions and preprocessing, and Bayesian Optimization. **Section 4** presents experimental setup, implementation details, baseline models, and evaluation metrics. **Section 5** discusses experimental results and comparative performance analysis of proposed model with baseline models. **Section 6** summarizes the contributions and Novelty of proposed approach. **Section 7** concludes the paper and suggests directions for future research.

2. Related Works

With the growing complexity of modern software systems, software defect prediction has become an active area of research for improving software quality. Numerous techniques, including statistical methods, ML, DL, ensemble learning, and hybrid models, have been investigated to enhance prediction accuracy and reliability.

2.1 Software Fault Prediction using ML

As software systems grow larger and more complex, traditional fault prediction approaches become less effective. Software fault prediction via supervised ML models have been proven to precisely predicting software bugs [3].

Experimental findings indicated that ML techniques effectively improved prediction accuracy and reduced software maintenance costs. Nevertheless, these techniques lacked advanced imbalance handling mechanisms and DL-based feature representation strategies. [4]. Comparative reliability prediction models have progressed the understanding of software system dependability [5].

In the software fault prediction domain, some ML techniques, like SVM, Naïve Bayes, ensemble learning, and feature engineering techniques, are performing well. These studies highlighted that feature selection and dimensionality reduction techniques were sound to boost classification accuracy [6-8].

Random forest combines multiple decision trees, which helps reduce overfitting and increase accuracy. It can work well with high-dimensional datasets. So, it can handle many input features [9]. It can also handle missing data without much pre-processing, unlike some other algorithms. It can provide feature importance scores, which can help us understand which features are most important for classification. Due to the above reasons, results delivered by using Random Forest are better than other classifiers.

In software defect prediction, several ML approaches like Decision Trees, Random Forests, Logistic Regression, Naïve Bayes, and SVM continue to serve as fundamental baselines. These approaches often provide competitive performance on structured software metrics. Their dependence on handcrafted features limits their capability to find complex nonlinear relationships and temporal software evolution patterns. This problem has encouraged the use of DL-based architectures.

2.2 Software Fault Prediction using DL

Deep learning-based approaches are effective in software fault prediction for complex software systems. These approaches substantially capture complex software metric patterns and outperform several traditional ML approaches [10].

ANN models generally outperform traditional ML techniques when sufficient training data are available. Class imbalance, feature redundancy, limited cross-project validation, and insufficient hybrid optimization techniques were identified. The authors recommended integrating optimization algorithms and feature engineering methods to improve prediction reliability [11].

2.3 Attention mechanism, Bayesian Optimization and Imbalance Handling Approaches

These ML and DL models are created and trained using labeled fault datasets. These datasets consisting of multiple independent variables like lines of codes, the complexity of the software, the size of the software, etc., and a dependent binary variable, either true or false. But the fault dataset may have some concerns like a class overlapping problem, class imbalance problem, null values etc. An imbalanced dataset is one in which one of the class data is present in the majority and another class data is present in the minority.

Attention mechanisms have recently emerged as an effective strategy for emphasizing fault-relevant software features while suppressing irrelevant information. Researchers demonstrated that attention-assisted recurrent architectures improve defect prediction performance by dynamically assigning importance weights to software attributes [12].

SMOTE was integrated with an ANN to address class imbalance problem in defect datasets. The SMOTE technique generated synthetic minority class samples, which demonstrated improved prediction accuracy, precision, recall, and F1-score compared with conventional ANN models. However, the study focused only on imbalance handling and did not investigate feature selection or model interpretability for further performance enhancement [13].

Models built using imbalanced datasets are biased. These biased models result in inaccurate predictions. Therefore, balancing the dataset is important. In this paper, various software fault prediction algorithms focusing on imbalance issue are discussed. According to the survey, SMOTE is the most commonly used data sampling technique for dealing with data imbalance issue [14].

Bayesian optimization is an efficient probabilistic approach that not only gets the optimal hyperparameters efficiently but also decreases the computation time. Bayesian optimization has been effective in optimizing model configuration. It finds the best hyperparameters. Convergence speed, prediction accuracy, and generalization performance were found to be more efficient than the manual tuning approaches [15, 16].

DL-enabled adaptive optimization technique for software fault prediction in Service-Oriented Architecture (SOA) is proposed. Experimental evaluation demonstrated superior performance over conventional ML methods in detecting service-oriented software faults. However, the approach primarily targeted SOA systems and did not evaluate its applicability across diverse software defect datasets [17].

Comprehensive surveys on imbalanced data handling have guided the development of robust software fault prediction techniques [18]. Research on dataset quality has emphasized its significant impact on accuracy, reliability and resilience of ML-driven software fault prediction models [19].

Preprocessing operations such as SMOTE, feature selection, and scaling should be applied only to the training partitions within each cross-validation fold to prevent data leakage and ensure reliable model evaluation. This methodology was adopted by applying SMOTE only to the training data and evaluating the models using Stratified K-Fold cross-validation [20].

2.4 Hybrid ML-DL and Ensemble Learning Approaches

A hybrid software defect prediction framework based on statistical analysis and ML algorithms was proposed in this research. The first step was to use statistical techniques to find the most important software metrics, and the second step was to use ML classification for defect prediction. The hybrid methodology improved prediction performance by eliminating irrelevant features and increasing the classification efficiency. The proposed model, however, did not include optimization algorithms or attention mechanisms for automatic feature weighting [21].

In this study, a hybrid ML-based software defect prediction framework was proposed. This proposed hybrid framework combines multiple classification techniques to improve defect detection accuracy. The proposed hybrid framework enhanced prediction performance by using effective preprocessing and integration of classifiers. The results indicated that the software quality assessment was improved than the single learning models. However, the study was not based on advanced optimization algorithms or deep feature attention mechanisms [22].

The innovative ensemble learning methods for software defect prediction using multiple ML classifiers were introduced. Ensemble learning combines multiple classifiers to improve predictive accuracy and generalization. Random Forest, Gradient Boosting, AdaBoost, and XGBoost are among the most frequently used ensemble algorithms in software defect prediction. This study demonstrated that ensemble learning effectively reduced prediction variance and enhanced defect identification as compared to individual classifiers. However, computational complexity, absence of intelligent feature attention mechanisms and their incapability to exploit deep feature learning capabilities available in neural networks remained important limitations [23-26].

Over the years, researchers have explored statistical models, ML algorithms, DL architectures, and hybrid approaches to improve prediction performance. There are a large number of performance measures available. The most popular measure is, Recall, ROC Area, Accuracy, F1-score, and Precision [27].

3. Research Methodology

3.1 Overall Framework of Proposed Hybrid Framework(RF-FA-ResANN)

The overall workflow of the proposed framework is illustrated in Figures 1 and 2, while Algorithm 1 describes the complete implementation.

3.2 Architecture of Proposed Hybrid Framework

Figure 1 illustrates the architecture of proposed framework. The architecture is composed of two complementary branches: Random Forest feature generation branch and Feature Attention-Based Residual ANN branch. The Random Forest feature generation branch is trained on the balanced training data and produces class probability scores, which are used to learn nonlinear relationships. The scores of these probabilities are added as features to the input features, which are then weighted by attention, creating an enriched feature representation. The augmented features are then fed into the Residual ANN, which uses feature attention to focus on informative software metrics and residual (skip) connections to enhance gradient propagation and feature learning. Then Bayesian Optimization is incorporated to automatically determine the optimal hyperparameters of the neural network, resulting in an efficient, robust, and generalized software defect prediction model.

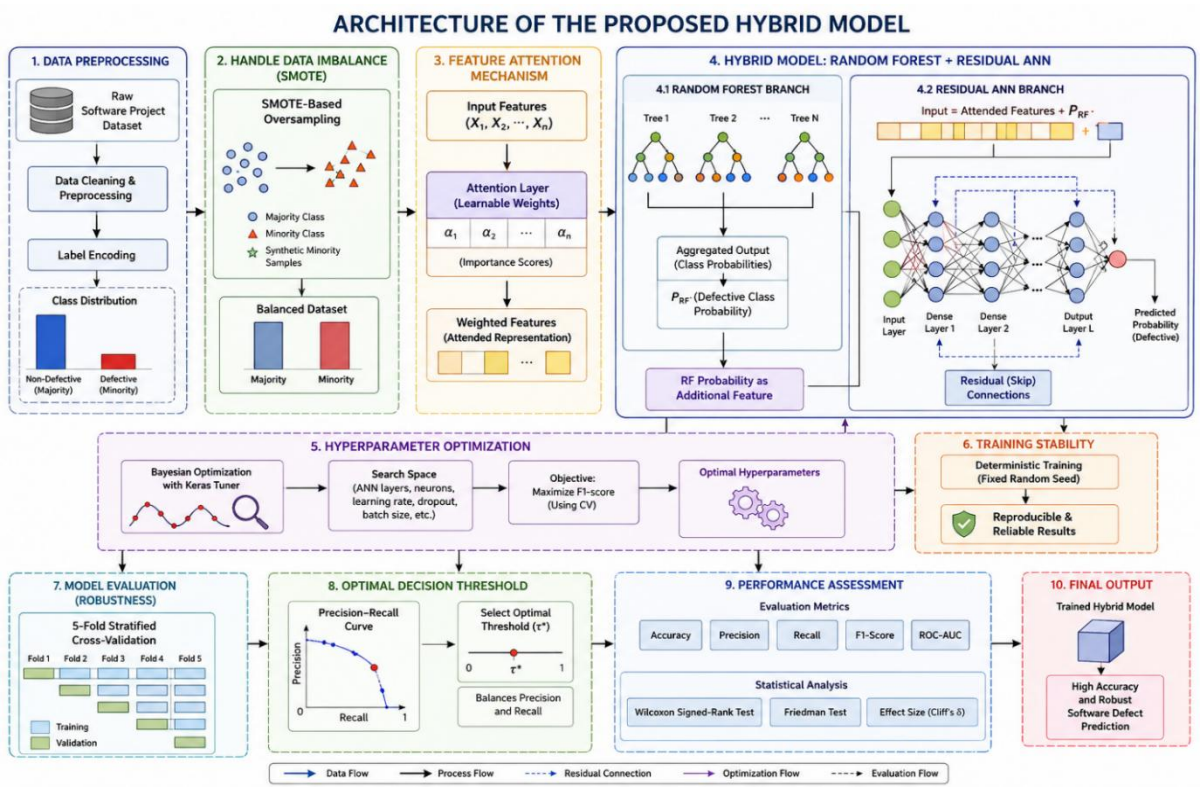


Figure 1. Architecture of the proposed hybrid framework

3.3 Process Flow of Proposed Hybrid Framework

Figure 2 presents the process flow of the proposed hybrid framework. After data preprocessing and class balancing, Random Forest generates probability-based features that are combined with the original features. The augmented feature set is classified using the Bayesian-optimized Feature Attention-based Residual ANN, and the model is evaluated using Stratified 5-Fold Cross-Validation and standard classification metrics.

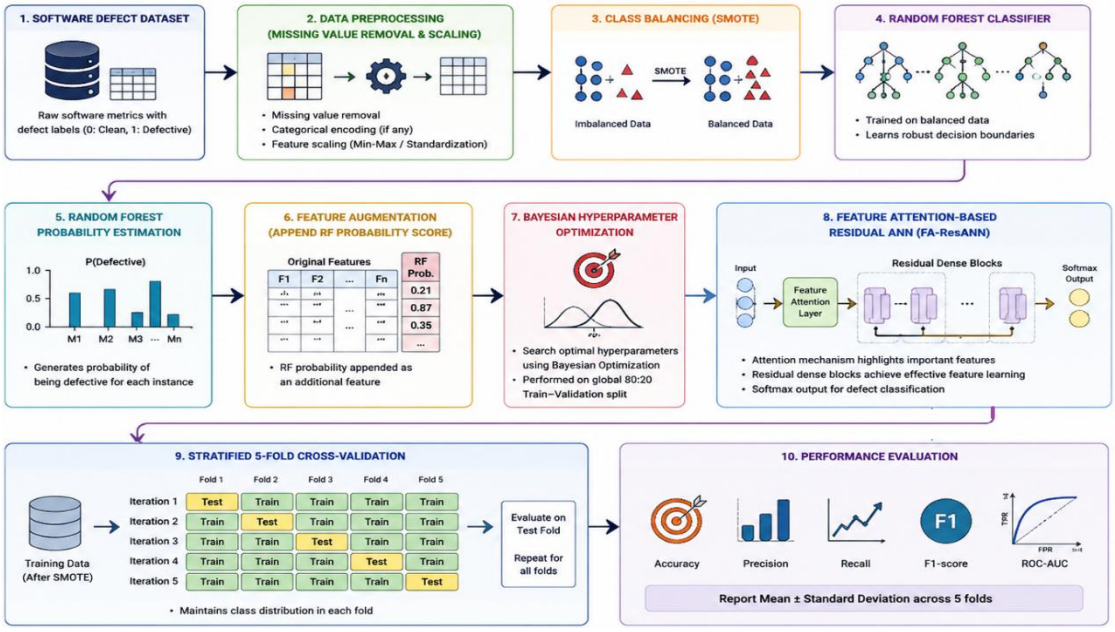


Figure 2. Process Flow of the Proposed framework

This workflow mirrors the steps carried out in proposed work. These are mainly preprocessing, feature augmentation with Random Forest, attention and residual ANN modelling, Bayesian tuning, and Cross-Validation evaluation.

Algorithm 1: Proposed Hybrid Random Forest–Feature Attention–Residual ANN (RF-FA-ResANN)

Input: Software defect dataset

$$D = \{X, Y\}, \quad ..(1)$$

Where

$$X = \{x_1, x_2, \dots, x_n\}$$

represents software metrics vector and

$$Y \in \{0,1\}$$

denotes defect labels.

Output: Optimized hybrid defect prediction model M^*.

Step 1: Load the software defect dataset D.

Step 2: Perform data preprocessing.

Remove missing values and standardize each feature using Z-score normalization:

$$Z_i = \frac{x_i - \mu}{\sigma} \quad ..(2)$$

where:

- μ= mean of the feature
- σ= standard deviation

Step 3: Within each training partition, apply SMOTE to the standardized training data to generate synthetic minority-class samples while leaving the corresponding validation partition unchanged.

Generate synthetic minority samples as

$$x_{new} = x_i + \lambda(x_{nn} - x_i), \quad 0 \leq \lambda \leq 1 \qquad ..(3)$$

where:

- x_i= minority sample,
- x_{nn}= nearest minority neighbour.

Obtained balanced dataset is

$$D_b = \{X_b, Y_b\}. \qquad ..(4)$$

Step 4: Train Random Forest classifier.

For each decision tree

$$T_i = f_i(X_b) \qquad i = 1,2, \dots \dots \dots \dots, N \qquad ..(5)$$

Where N is the total number of trees.

The Random Forest prediction probability is

$$P_{RF}(x) = \frac{1}{N}\sum_{i=1}^{N} T_i\,(x). \qquad ..(6)$$

Step 5: Construct augmented feature vector

Append Random Forest probability to the original feature vector:

$$X' = [X_b,\, P_{RF}(x)]. \qquad ..(7)$$

Step 6: Compute feature attention weights

Attention weights are computed as

$$A = \sigma(W_a X' + b_a), \qquad ..(8)$$

where

- W_a= attention weight matrix
- b_a= bias vector
- $\sigma(\cdot)$= sigmoid activation function

Generate weighted features

$$X_A = A \odot X', \qquad ..(9)$$

where $\odot$denotes element-wise multiplication.

Step 7: Residual Dense Block

Hidden representation is computed as

$$H = ReLU(W_h X_A + b_h), \qquad ..(10)$$

Residual connection

$$R = H + X_A. \qquad ..(11)$$

Step 8: ANN Output Layer.

Compute the predicted class probabilities using the Softmax activation function:

$$\hat{y} = \text{Softmax}(z)\,, \quad ..(12)$$

Train the RF-FA-ResANN model by minimizing the Sparse Categorical Cross-Entropy loss:

This will render as:

$$L = -\frac{1}{m}\sum_{i=1}^{m} log\left(p_{i,y_i}\right) \quad ..(13)$$

Where

m= Batch size (number of samples processed in one training iteration).

p_{i,y_i}= Predicted probability of the true class for sample i.

Step 9: Bayesian Hyperparameter Optimization.

Find optimal hyperparameter set

$$\theta^* = \arg\max_{\theta\epsilon\Omega}\ f(\theta), \quad ..(14)$$

where

$$\theta = \{\eta,\ H,\ Dropout,\ Batch\ Size,\ Epochs\},$$

Ω is the hyperparameter search space and $f(\theta)$is the Cross-Validation objective function.

Step 10: Perform Stratified 5-Fold Cross-Validation.

Split balanced dataset into

$$D_b = \{F_1, F_2, F_3, F_4, F_5\}.$$

Train the Random Forest and RF-FA-ResANN model using the SMOTE-balanced training folds, and evaluate on the untouched validation fold.

Average performance

$$Score = \frac{1}{5}\sum_{i=1}^{5}\ Score_i. \quad ..(15)$$

Step 11: Compute evaluation metrics.

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad ..(16)$$

Precision

$$Precision = \frac{TP}{TP + FP} \quad ..(17)$$

Recall

$$Recall = \frac{TP}{TP + FN} \quad ..(18)$$

F1-score

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad ..(19)$$

ROC-AUC

$$ROC - AUC = \int_0^1 TPR(FPR)\, d(FPR) \quad ..(20)$$

3.4 Dataset Description and Dataset Preprocessing

The research methodology described in Figure 1 follows a systematic process of software fault prediction. We used the following two different datasets for the evaluation of these eight models.

(i) Software_defect Dataset

First dataset used in this research was a Software_defect dataset, accessible at https://coredaojournal.com/coreda/index.php/home/announcement. The software_defect.csv (original file) contains 100,885 records with 23 attributes, where each record represents a software module and each attribute corresponds to a static code metric or the defect label. Out of 23 attributes, 22 attributes represent software engineering metrics and one attribute represents the defect class. Each instance corresponds to a software module, and the target variable indicates whether the module is defective or non-defective. The dataset includes a combination of McCabe complexity metrics, Halstead software science metrics, and size-related software metrics like Lines of Code (LOC) etc., that characterize the structural complexity, maintainability, and development characteristics of source code. These software metrics collectively describe the internal characteristics of software modules and are widely recognized as indicators of software quality and fault proneness.

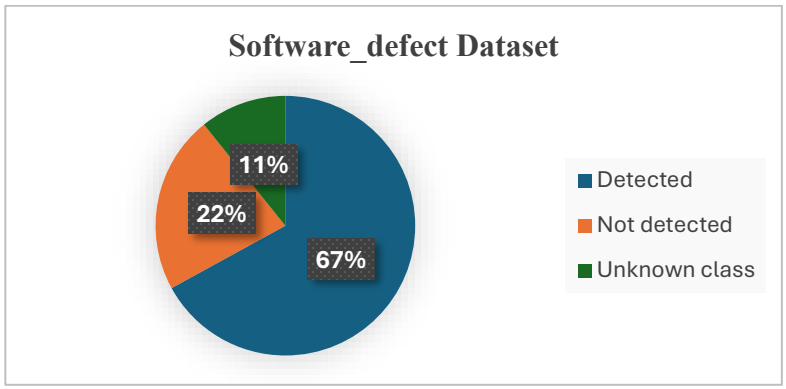


Figure 3. Class Imbalance Ratio of Software_defect Dataset

(ii) NASA JM1 Dataset

Second dataset used for experimental evaluation of the proposed model is NASA JM1 dataset. This dataset is accessible at https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction?select=jm1.csv. The dataset is based on the well-known NASA PROMISE repository, specifically the NASA JM1 project, which has been extensively utilized by researchers for evaluating software quality prediction models. The repository includes software metrics from real software modules and their associated defect labels, which is suitable for software engineering binary classification problems. The NASA JM1 data set contains 10,885 software modules with 22 attributes, 21 of which are software engineering metrics and one attribute is the defect class. This dataset also includes same type of attributes as software_defect dataset.

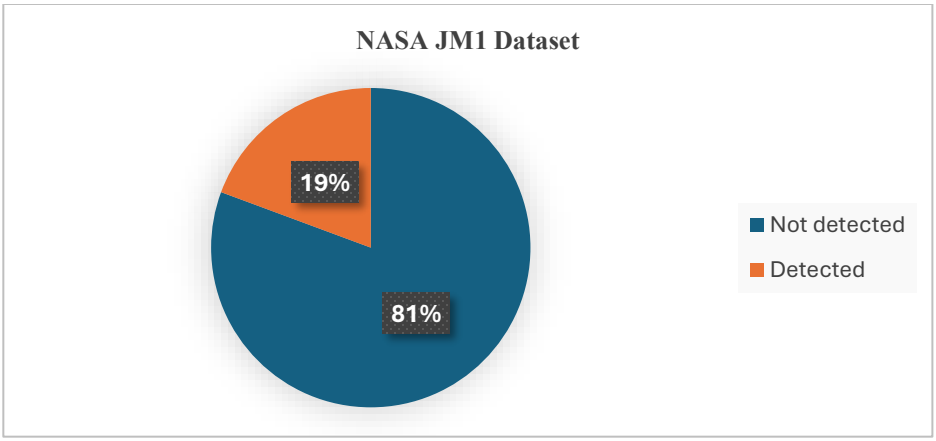


Figure 4. Class Imbalance Ratio of NASA JM1 Dataset

(iii) Dataset Preprocessing for both datasets

Both datasets are proposed to follow the same common framework starting with dataset preprocessing. The data is loaded from a CSV file, with the aim of identifying and excluding missing, inconsistent, invalid, and redundant records to maintain data integrity. Non-informative features are also eliminated to further reduce dimensionality and computation. The cleaned data is then exported to a new CSV file. This CSV file will be used for further processing, such as normalization, feature selection, training models, and assessing their performance. These pre-processing steps enhance the efficiency and prediction accuracy of different software defect prediction models.

The software_ defect dataset (Dataset 1) is preprocessed and then exported as cleaneddatasetsdd.csv (a new csv). This file has 31750 modules, with 17 attributes, 16 of which are software engineering metrics and 1 is the defect class. Likewise, once preprocessed, NASA JM1 dataset (Dataset 2) is exported as cleaneddatasetjm1.csv (a new csv). This file contains 10885 modules each having equal and same attributes as in cleaneddatasetsdd.csv.

3.5 Residual Artificial Neural Network (ResANN)

In proposed framework, ResANN consists of multiple fully connected (dense) layers with ReLU activation to learn complex nonlinear relationships among software metrics. To overcome gradient degradation and facilitate deeper feature learning, skip (residual) connections are incorporated by adding the transformed input features to the output of the dense layers. This residual learning strategy preserves important feature information, improves gradient propagation during backpropagation, accelerates convergence, and enhances overall predictive performance of the model.

3.6 Bayesian Optimization

Bayesian hyperparameter optimization was performed once prior to model evaluation using an independent 80:20 stratified train-validation split. During this tuning stage, data preprocessing, feature standardization, SMOTE oversampling, Random Forest probability-based feature augmentation, and ANN training were applied only to the training partition, while the validation partition remained untouched. The optimal hyperparameters identified during this stage were fixed and subsequently used for Stratified 5-Fold Cross-Validation. During Stratified 5-Fold Cross-Validation, StandardScaler, SMOTE, Random Forest training, and Random Forest probability-based feature augmentation were performed independently within each training fold, while the corresponding validation fold remained completely untouched to prevent data leakage and ensure an unbiased evaluation.

(i) Bayesian Optimization Search Space

The search space defines the range of hyperparameters explored during Bayesian Optimization. Based on the proposed RF-FA-ResANN architecture, Keras Tuner searches for the optimal configuration of the hidden layers, dropout rate, and optimizer learning rate.

Table 1. Hyperparameter Search Space

Hyperparameter	Search Type	Search Range
First Hidden Layer Neurons (dense_1)	Integer	64 – 256 (step size = 64)
Second Hidden Layer Neurons (dense_2)	Integer	32 – 128 (step size = 32)
Dropout Rate	Float	0.20 – 0.50 (step size = 0.05)
Adam Learning Rate	Logarithmic Float	(1 \times 10^{-4}) to (1 \times 10^{-2})

The search space was selected to provide sufficient flexibility for learning complex nonlinear relationships while controlling model complexity and reducing overfitting.

(ii) Objective Function

The objective function guides the Bayesian Optimization process by evaluating the quality of each hyperparameter configuration. In this research, the optimization objective is to maximize the validation accuracy (val_accuracy) obtained during model training. For every trial, the candidate FA-ResANN model is trained for 30 epochs using the augmented training dataset, while an Early Stopping mechanism monitors the validation loss. Training is terminated if the validation loss does not improve for eight consecutive epochs, and the model weights corresponding to the best validation performance are automatically restored. Mathematically, the optimization problem can be expressed as shown in equation 14.

The optimal hyperparameters obtained during this phase are subsequently used for all folds of the Stratified 5-Fold Cross-Validation.

(iii) Keras Tuner Implementation

Bayesian Optimization is implemented using the BayesianOptimization tuner provided by the Keras Tuner library. The tuner automatically generates multiple candidate FA-ResANN architectures by sampling different combinations of hyperparameters from the predefined search space.

3.7 Model Training

The proposed RF-FA-ResANN model was trained using the Adam optimizer with the Sparse Categorical Cross-Entropy loss function, which is suitable for multi-class classification with integer-encoded labels. The model was trained for a maximum of 30 epochs with a batch size of 128. To prevent overfitting, Early Stopping was employed by monitoring the validation loss with a patience of 8 epochs, restoring the best-performing model weights.

3.8 Evaluation Strategy

The performance of the proposed model was evaluated using Stratified 5-Fold Cross-Validation. It ensures that class distribution was preserved across all folds for a fair and robust performance assessment. For each fold, the optimal classification threshold was determined using the Precision–Recall curve by selecting the threshold that maximized the F1-score, providing a balanced trade-off between precision and recall.

4. Experimental Setup

4.1 Experimental Environment

The experiments were conducted using Google Colab platform. Furthermore, all models were trained and accessed on the same hardware and software configurations. It ensured consistent and fair experimental conditions. All the models were trained and evaluated for both datasets.

4.2 Hyperparameter Settings

The performance of the proposed model is influenced by several network and training hyperparameters. Bayesian Optimization was employed to obtain an optimal model configuration, using the Keras Tuner framework.

Multiple combinations of hyperparameters were explored During the optimization process. The configuration that achieved the highest validation accuracy was selected for the final model.

Table 2. The hyperparameter settings used in the research

Parameter	Value
Optimizer	Adam
Loss Function	Sparse Categorical Cross-Entropy
Output Activation	Softmax
Hidden Layer 1 Neurons (dense_1)	Optimized using Bayesian Optimization (64–256). *Optimum value-192*
Hidden Layer 2 Neurons (dense_2)	Optimized using Bayesian Optimization (32–128). *Optimum value -64*
Hidden Layer Activation	ReLU
Feature Attention Activation	Sigmoid
Residual Shortcut Activation	Linear
Dropout Rate	Optimized using Bayesian Optimization (0.20–0.50).*Optimum value - 0.30*
Learning Rate	Optimized using Bayesian Optimization (1×10^{-4} – 1×10^{-2}).*Optimum value -0.0012*
Batch Size	128
Maximum Epochs	30
Early Stopping	Enabled
Early Stopping Monitor	Validation Loss
Patience	8 Epochs
Random Forest Trees	100
Random Forest Probability	Appended as an additional input feature
SMOTE	Enabled
Cross-Validation	Stratified 5-Fold
Hyperparameter Optimization	Bayesian Optimization (Keras Tuner)
Objective Function	Validation Accuracy (val_accuracy)
Maximum Trials	2
Executions per Trial	1
Random Seed	42

4.3 Evaluation Metrics

To ensure a fair and unbiased comparison, all models are developed and evaluated on both datasets. Their performance is assessed using Accuracy, Recall, Precision, F1-Score, and ROC-AUC [28].

ROC-AUC: It measures model's ability to distinguish between defective and non-defective software modules across different classification thresholds.

Accuracy: It measures overall percentage of correct predictions.

Precision: It represents, accuracy of positive predictions only.

Recall: It represents, percentage of actual positives caught.

F1-Score: It represents balanced average of precision and recall.

4.4 Baseline Models

In this research work, seven ML and DL based baseline models are evaluated for software fault prediction. Key objective of this evaluation is to compare the prediction capability, robustness, and generalization performance of these models with proposed model.

Table 3. List of baseline models evaluated in the research

Sr. No.	Model	Category	Purpose
1	Support Vector Machine (SVM)	ML	Classical margin-based classifier
2	Naïve Bayes (NB)	ML	Probabilistic baseline classifier
3	Random Forest (RF)	ML	Ensemble learning baseline
4	ANN	DL	Basic neural network model
5	Feature Attention-Based ANN(FA-ANN)	DL	Evaluates the effect of feature attention
6	Residual Artificial Neural Network (ResANN)	DL	Evaluates the effect of residual learning
7	Feature Attention-Based Residual ANN (FA-ResANN)	DL	Evaluates the combined effect of attention and residual learning

5. Results

5.1 Naïve Bayes Model

The Gaussian Naïve Bayes (NB) classifier was evaluated using the common experimental framework as described in Section 4. Table 4 and Figure 5 present the predictive performance of the Naïve Bayes (NB) model on the Software_defect and NASA JM1 datasets.

Software_defect Dataset	NASA JM1 Dataset
>>> Running Stratified Cross-Validation using Naive Bayes... ========== CROSS-VALIDATION FOLD 1/5 ========== [Fold 1] Optimized Threshold Used: 0.0001 >> Fold 1 Metrics -> Accuracy: 0.6047 | Precision: 0.1716 | Recall: 0.7972 | F1-Score: 0.2824 ========== CROSS-VALIDATION FOLD 2/5 ========== [Fold 2] Optimized Threshold Used: 0.7029 >> Fold 2 Metrics -> Accuracy: 0.8503 | Precision: 0.2899 | Recall: 0.3683 | F1-Score: 0.3244 ========== CROSS-VALIDATION FOLD 3/5 ========== [Fold 3] Optimized Threshold Used: 0.0073 >> Fold 3 Metrics -> Accuracy: 0.7832 | Precision: 0.2252 | Recall: 0.5007 | F1-Score: 0.3107 ========== CROSS-VALIDATION FOLD 4/5 ========== [Fold 4] Optimized Threshold Used: 0.0097 >> Fold 4 Metrics -> Accuracy: 0.7816 | Precision: 0.2322 | Recall: 0.5358 | F1-Score: 0.3240 ========== CROSS-VALIDATION FOLD 5/5 ========== [Fold 5] Optimized Threshold Used: 0.6565 >> Fold 5 Metrics -> Accuracy: 0.8521 | Precision: 0.2907 | Recall: 0.3567 | F1-Score: 0.3203 == --- Evaluation Summary (Gaussian Naive Bayes Model) --- Accuracy: 0.7744 (± 0.0902) Precision: 0.2419 (± 0.0447) Recall: 0.5118 (± 0.1593) F1-Score: 0.3124 (± 0.0158) Mean AUC: 0.73948	>>> Running Stratified Cross-Validation using Naive Bayes... ========== CROSS-VALIDATION FOLD 1/5 ========== [Fold 1] Optimized Threshold Used: 0.0000 >> Fold 1 Metrics -> Accuracy: 0.6417 | Precision: 0.5334 | Recall: 0.5879 | F1-Score: 0.5593 ========== CROSS-VALIDATION FOLD 2/5 ========== [Fold 2] Optimized Threshold Used: 0.0000 >> Fold 2 Metrics -> Accuracy: 0.6500 | Precision: 0.5422 | Recall: 0.6105 | F1-Score: 0.5743 ========== CROSS-VALIDATION FOLD 3/5 ========== [Fold 3] Optimized Threshold Used: 0.0000 >> Fold 3 Metrics -> Accuracy: 0.6665 | Precision: 0.5620 | Recall: 0.6247 | F1-Score: 0.5917 ========== CROSS-VALIDATION FOLD 4/5 ========== [Fold 4] Optimized Threshold Used: 0.0000 >> Fold 4 Metrics -> Accuracy: 0.6477 | Precision: 0.5385 | Recall: 0.6299 | F1-Score: 0.5806 ========== CROSS-VALIDATION FOLD 5/5 ========== [Fold 5] Optimized Threshold Used: 0.0000 >> Fold 5 Metrics -> Accuracy: 0.6537 | Precision: 0.5466 | Recall: 0.6192 | F1-Score: 0.5806 == --- Evaluation Summary (Gaussian Naive Bayes Model) --- Accuracy: 0.6519 (± 0.0083) Precision: 0.5445 (± 0.0097) Recall: 0.6144 (± 0.0148) F1-Score: 0.5773 (± 0.0106) Mean AUC: 0.67024

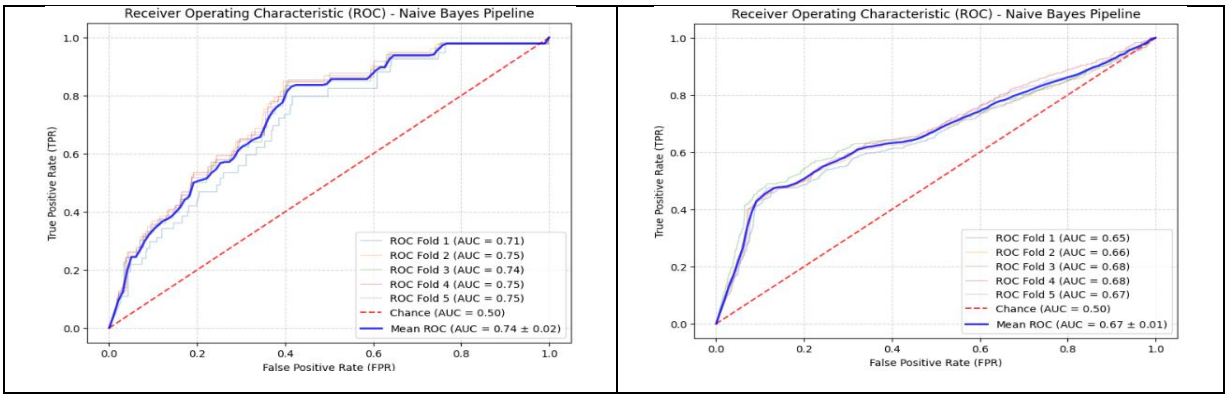


Figure 5. Predictive performance of Naïve Bayes model for Software_defect dataset & JM1 dataset

5.2 Support Vector Machine (SVM) model

SVM classifier was also evaluated using the same experimental framework adopted for all baseline models. The classification performance obtained on both datasets is summarized in Figure 6 and Table 4.

Software_defect Dataset

```
>>> Running Stratified Cross-Validation using Support Vector Machine (SVM)...

==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.9211
>> Fold 1 Metrics -> Accuracy: 0.9591 | Precision: 0.8100 | Recall: 0.7586 | F1-Score: 0.7835

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 0.9129
>> Fold 2 Metrics -> Accuracy: 0.9657 | Precision: 0.8176 | Recall: 0.8345 | F1-Score: 0.8259

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.9187
>> Fold 3 Metrics -> Accuracy: 0.9553 | Precision: 0.7610 | Recall: 0.7903 | F1-Score: 0.7754

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.9150
>> Fold 4 Metrics -> Accuracy: 0.9641 | Precision: 0.8064 | Recall: 0.8320 | F1-Score: 0.8190

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 0.8224
>> Fold 5 Metrics -> Accuracy: 0.9498 | Precision: 0.6848 | Recall: 0.9008 | F1-Score: 0.7781

============================================================
--- Evaluation Summary (Support Vector Machine Model) ---
Accuracy:  0.9588  (± 0.0058)
Precision: 0.7760  (± 0.0497)
Recall:    0.8232  (± 0.0479)
F1-Score:  0.7964  (± 0.0216)
Mean AUC:  0.95239
```

NASA JM1 Dataset

```
>>> Running Stratified Cross-Validation using Support Vector Machine (SVM)...

==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.3040
>> Fold 1 Metrics -> Accuracy: 0.6610 | Precision: 0.5392 | Recall: 0.8504 | F1-Score: 0.6599

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 0.2964
>> Fold 2 Metrics -> Accuracy: 0.6481 | Precision: 0.5272 | Recall: 0.8741 | F1-Score: 0.6577

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.3443
>> Fold 3 Metrics -> Accuracy: 0.6936 | Precision: 0.5733 | Recall: 0.8124 | F1-Score: 0.6722

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.3810
>> Fold 4 Metrics -> Accuracy: 0.6890 | Precision: 0.5724 | Recall: 0.7782 | F1-Score: 0.6596

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 0.3565
>> Fold 5 Metrics -> Accuracy: 0.6844 | Precision: 0.5646 | Recall: 0.8090 | F1-Score: 0.6650

============================================================
--- Evaluation Summary (Support Vector Machine Model) ---
Accuracy:  0.6752  (± 0.0176)
Precision: 0.5553  (± 0.0187)
Recall:    0.8248  (± 0.0337)
F1-Score:  0.6629  (± 0.0053)
Mean AUC:  0.78084
```

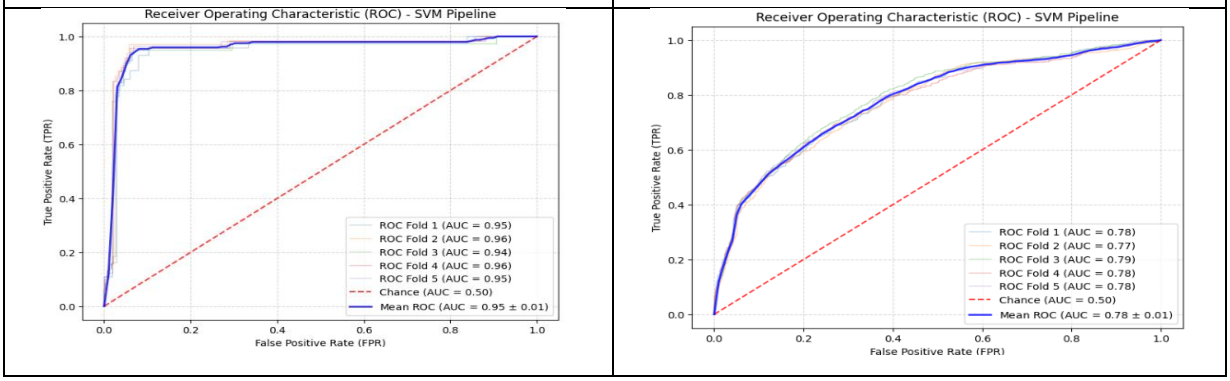


Figure 6. Predictive performance of SVM model for Software_defect dataset & JM1 dataset

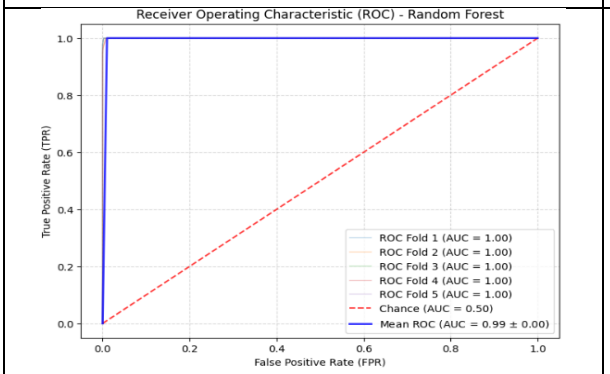
5.3 Random Forest Model(RF)

RF classifier was trained and evaluated under the same experimental settings as described in Section 4. The obtained performance metrics for both datasets are reported in the Table 4 and Figure 7.

Software_defect Dataset	NASA JM1 Dataset

```
>>> Phase 1: Initializing Baseline Random Forest Configuration...

>>> Phase 2: Running Stratified Cross-Validation on Random Forest...

==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.8980
>> Fold 1 Metrics -> Accuracy: 0.9981 | Precision: 0.9811 | Recall: 1.0000 | F1-Score: 0.9904

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 1.0000
>> Fold 2 Metrics -> Accuracy: 0.9961 | Precision: 1.0000 | Recall: 0.9600 | F1-Score: 0.9796

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.9062
>> Fold 3 Metrics -> Accuracy: 0.9977 | Precision: 0.9771 | Recall: 1.0000 | F1-Score: 0.9884

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.8941
>> Fold 4 Metrics -> Accuracy: 0.9961 | Precision: 0.9874 | Recall: 0.9725 | F1-Score: 0.9799

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 1.0000
>> Fold 5 Metrics -> Accuracy: 0.9984 | Precision: 1.0000 | Recall: 0.9835 | F1-Score: 0.9917

============================================================
--- Enhanced Evaluation Summary (Random Forest) ---
Accuracy:  0.9973  (± 0.0010)
Precision: 0.9891  (± 0.0095)
Recall:    0.9832  (± 0.0156)
F1-Score:  0.9860  (± 0.0052)
Mean AUC:  0.99993
```

```
==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.6400
>> Fold 1 Metrics -> Accuracy: 0.8875 | Precision: 0.8443 | Recall: 0.8694 | F1-Score: 0.8566

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 0.7000
>> Fold 2 Metrics -> Accuracy: 0.8797 | Precision: 0.8690 | Recall: 0.8112 | F1-Score: 0.8391

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.6300
>> Fold 3 Metrics -> Accuracy: 0.9003 | Precision: 0.8539 | Recall: 0.8955 | F1-Score: 0.8742

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.6278
>> Fold 4 Metrics -> Accuracy: 0.8746 | Precision: 0.8361 | Recall: 0.8410 | F1-Score: 0.8386

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 0.6500
>> Fold 5 Metrics -> Accuracy: 0.8865 | Precision: 0.8522 | Recall: 0.8553 | F1-Score: 0.8538

============================================================
--- Enhanced Evaluation Summary (Random Forest) ---
Accuracy:  0.8857  (± 0.0087)
Precision: 0.8511  (± 0.0110)
Recall:    0.8545  (± 0.0281)
F1-Score:  0.8524  (± 0.0131)
Mean AUC:  0.92865
```



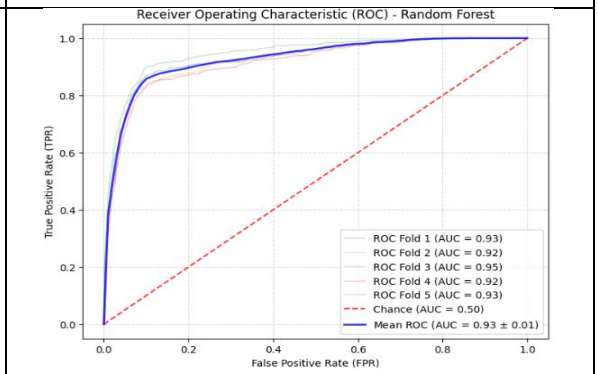


Figure 7. Predictive performance of Random Forest model for Software_defect dataset & JM1 dataset

5.4 Artificial Neural Network (ANN) Model

The Artificial Neural Network (ANN) model was evaluated using the common preprocessing pipeline, Bayesian-optimized training procedure, and Stratified 5-Fold Cross-Validation. Its performance on both datasets is presented in the Table 4 and Figure 8.

Software_defect Dataset	NASA JM1 Dataset

```
==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.9936
>> Fold 1 Metrics -> Accuracy: 0.9939 | Precision: 0.9802 | Recall: 0.9572 | F1-Score: 0.9686

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 0.9961
>> Fold 2 Metrics -> Accuracy: 0.9948 | Precision: 1.0000 | Recall: 0.9462 | F1-Score: 0.9724

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.9946
>> Fold 3 Metrics -> Accuracy: 0.9933 | Precision: 1.0000 | Recall: 0.9310 | F1-Score: 0.9643

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.9967
>> Fold 4 Metrics -> Accuracy: 0.9953 | Precision: 1.0000 | Recall: 0.9518 | F1-Score: 0.9753

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 0.9961
>> Fold 5 Metrics -> Accuracy: 0.9946 | Precision: 0.9804 | Recall: 0.9642 | F1-Score: 0.9722

============================================================
--- Enhanced Evaluation Summary (ANN Model) ---
Accuracy:  0.9944  (± 0.0007)
Precision: 0.9921  (± 0.0096)
Recall:    0.9501  (± 0.0112)
F1-Score:  0.9706  (± 0.0038)
Mean AUC:  0.99928
```

```
==================== CROSS-VALIDATION FOLD 1/5 ====================
[Fold 1] Optimized Threshold Used: 0.4352
>> Fold 1 Metrics -> Accuracy: 0.6886 | Precision: 0.5726 | Recall: 0.7684 | F1-Score: 0.6562

==================== CROSS-VALIDATION FOLD 2/5 ====================
[Fold 2] Optimized Threshold Used: 0.4237
>> Fold 2 Metrics -> Accuracy: 0.6748 | Precision: 0.5556 | Recall: 0.7945 | F1-Score: 0.6540

==================== CROSS-VALIDATION FOLD 3/5 ====================
[Fold 3] Optimized Threshold Used: 0.4379
>> Fold 3 Metrics -> Accuracy: 0.6959 | Precision: 0.5758 | Recall: 0.8124 | F1-Score: 0.6739

==================== CROSS-VALIDATION FOLD 4/5 ====================
[Fold 4] Optimized Threshold Used: 0.4646
>> Fold 4 Metrics -> Accuracy: 0.6904 | Precision: 0.5755 | Recall: 0.7639 | F1-Score: 0.6565

==================== CROSS-VALIDATION FOLD 5/5 ====================
[Fold 5] Optimized Threshold Used: 0.4428
>> Fold 5 Metrics -> Accuracy: 0.6849 | Precision: 0.5667 | Recall: 0.7912 | F1-Score: 0.6604

============================================================
--- Enhanced Evaluation Summary (ANN Model) ---
Accuracy:  0.6869  (± 0.0070)
Precision: 0.5692  (± 0.0075)
Recall:    0.7861  (± 0.0178)
F1-Score:  0.6602  (± 0.0072)
Mean AUC:  0.78902
```

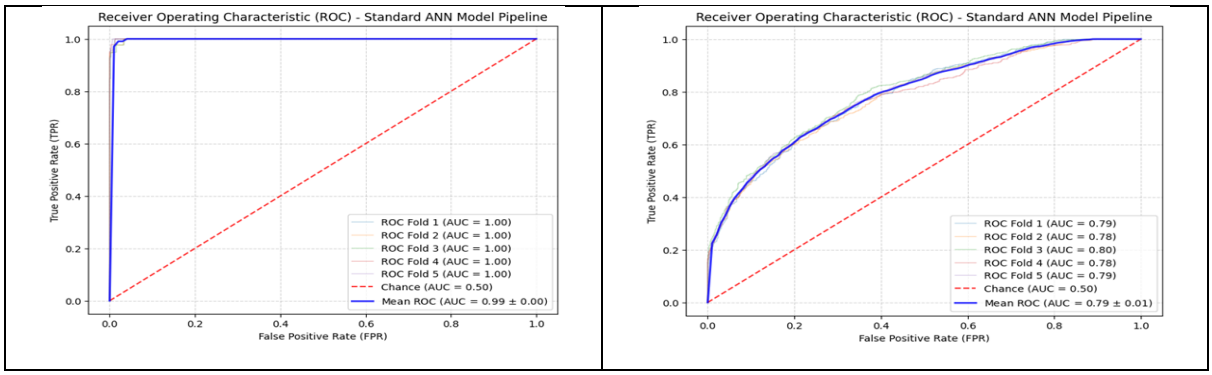


Figure 8. Predictive performance of ANN model for Software_defect dataset & JM1 dataset

5.5 Feature Attention based Artificial Neural Network Model

The Feature Attention-Based ANN model was evaluated using the same experimental framework as the baseline models. The classification results obtained on both datasets are presented in the corresponding Table 4 and Figure 9.

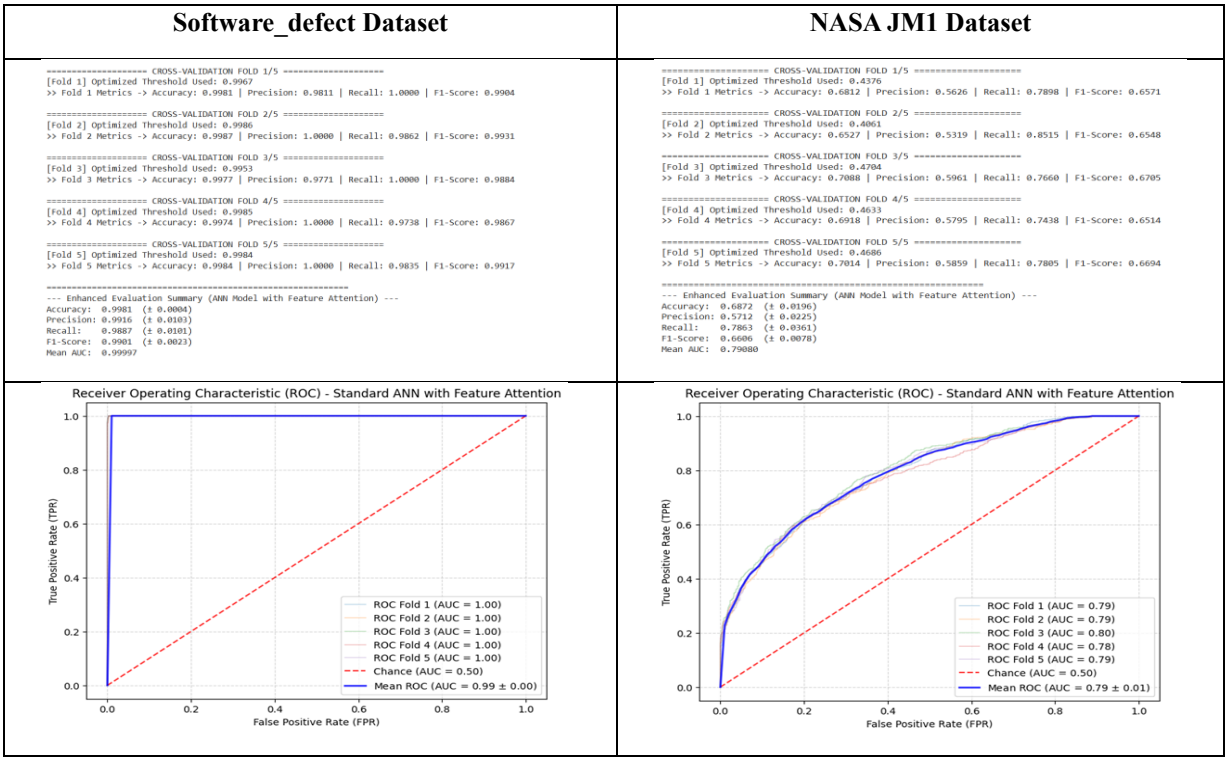


Figure 9. Predictive performance of Feature Attention-Based ANN for Software_defect dataset & JM1 dataset

5.6 Residual Artificial Neural Network (Res ANN) Model

The Residual Artificial Neural Network (ResANN) model was evaluated using the common experimental framework described in Section 4. The predictive performance achieved on both datasets is summarized in the Table 4 and Figure 10.

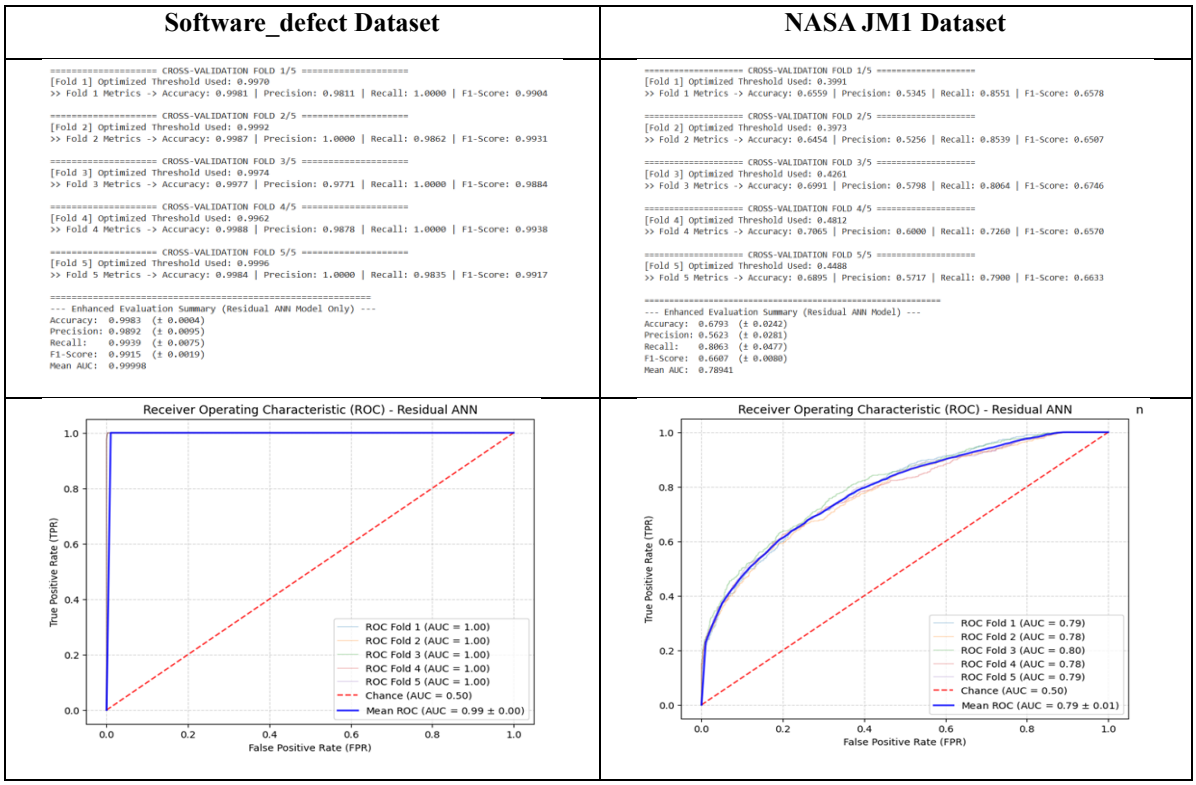


Figure 10. Predictive performance of Residual ANN (ResANN) for Software_defect dataset & JM1 dataset

5.7 Feature Attention based Residual Artificial Neural Network (FA-Res ANN) Model

The Feature Attention-Based Residual ANN (FA-ResANN) model was evaluated using the same preprocessing, training, and evaluation strategy adopted throughout this study. The experimental results for both datasets are presented in the corresponding Table 4 and Figure 11.

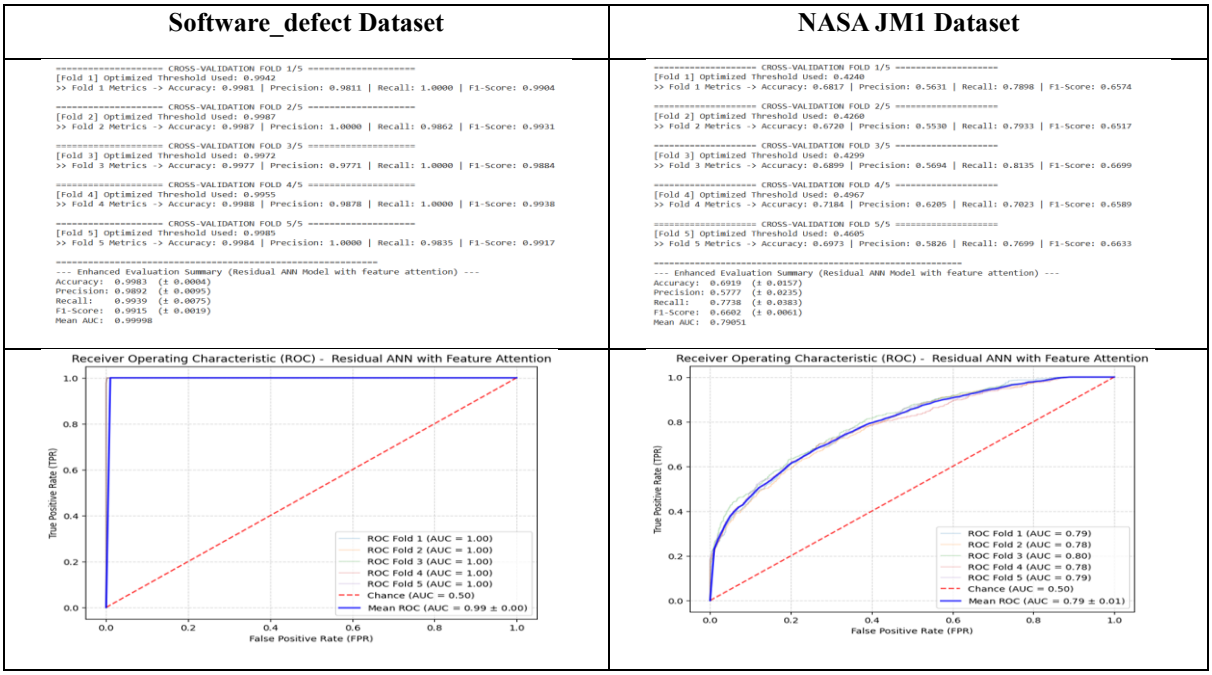


Figure 11. Predictive performance of Feature Attention-Based ResANN for Software_defect dataset & JM1 dataset

5.8 Hybrid Random Forest with Feature Attention and Residual ANN (RF-FA-ResANN) Model

The proposed Hybrid RF–FA-ResANN model was evaluated using the common experimental framework. Performance on the Software_defect and JM1 datasets is presented in the Figure 12 and Table 4.

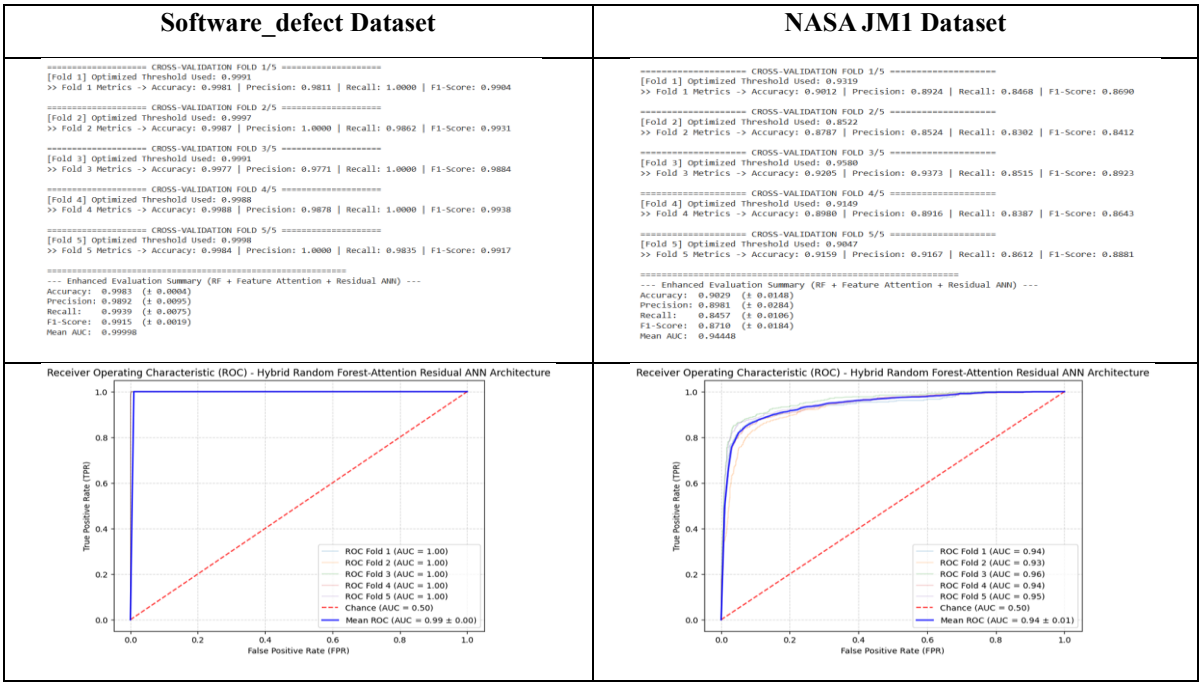


Figure 12. Predictive performance of Proposed model for Software_defect dataset & JM1 dataset

5.9 Comparative Analysis

Table 4 shows the comparative performance of the proposed RF-FA-ResANN framework with seven baseline models. This comparison was carried out on two widely available Software_defect datasets (NASA JM1 and Software_defect) and the five widely used performance metrics, Accuracy, Precision, Recall, F1-score, and ROC-AUC.

The baseline models had moderate to good classification performance for NASA JM1. The Naïve Bayes classifier had the least accuracy (65.19%), F1-score (57.73%), and ROC-AUC (0.67024), which means it is not very good at distinguishing between defective and non-defective software modules. SVM model achieved an accuracy of 67.52%, the recall of 82.48%, while precision was 55.53%, which indicated that there are more false positive predictions. The standalone ANN, ANN with Feature Attention, Residual ANN, and Residual ANN with Feature Attention models achieved comparable results, with the accuracy ranging from 68.69% to 69.19%, F1-scores close to 66%, and AUC values near to 0.79. The incorporation of Feature Attention and residual connections alone produced only marginal improvements over the standard ANN. It indicates that deep learning architectures without enriched feature representations were unable to effectively exploit the characteristics of the NASA JM1 dataset. The Random Forest classifier proved to be much more effective than the other classifiers, showing accuracy of 88.57%, precision of 85.11%, recall of 85.45%, F1-score of 85.24%, and ROC-AUC of 0.92865, indicating that it was able to capture more complex decision boundaries.

The proposed RF-FA-ResANN achieved the best overall performance with an accuracy of 90.29%, precision of 89.81%, recall of 84.57%, F1-score of 87.10%, and ROC-AUC of 0.94448. Compared with the strongest baseline (Random Forest), the proposed model improved the accuracy by 1.72%, precision by 4.70%, F1-score by 1.86%, and ROC-AUC by 1.71%, while maintaining competitive recall. Overall, the proposed model provides the most balanced and robust prediction performance on the challenging JM1 dataset.

The same phenomenon was observed in the Software_defect dataset. Random Forest had the best performance among the conventional ML models with an accuracy of 99.73%, precision of 98.91%, recall of 98.32%, F1-score of 98.60% and ROC-AUC of 0.99993. SVM and Naïve Bayes, on the other hand, achieved significantly lower results, especially Naïve Bayes with an F1-score of 31.24%, which is poor performance in dealing with the dataset. The ANN

model with standard architecture was able to reach an accuracy of 99.44% and an F1-score of 97.06% among the deep learning models. The Feature Attention mechanism boosted the accuracy to 99.81% and the F1-score to 99.01%, showing the value of giving adaptive weights to software metrics. Likewise, the Residual ANN achieved an accuracy of 99.83% and F1-score of 99.15%, demonstrating that the residual connections can enhance feature propagation and learning stability. The ResANN with Feature Attention performed equally well, indicating that the dataset was already close to the optimal classification performance.

The proposed RF-FA-ResANN achieved an accuracy of 99.83%, precision of 98.92%, recall of 99.39%, F1-score of 99.15%, and ROC-AUC of 0.99998 on Software_defect dataset. Its performance was comparable to the best-performing deep learning models, including ResANN and ResANN with Feature Attention, while also achieving a slightly higher ROC-AUC than Random Forest. These results indicate that the proposed model provides highly accurate and reliable software defect prediction.

Table 4. Comparative Analysis of all models for NASA JM1 Dataset & Software_defect Dataset

	Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
NASA JM1 Dataset	**SVM**	0.6752	0.5553	0.8248	0.6629	0.78084
	NAÏVE Bayes	0.6519	0.5445	0.6144	0.5773	0.67024
	Random Forest	0.8857	0.8511	0.8545	0.8524	0.92865
	ANN	0.6869	0.5692	0.7861	0.6602	0.78902
	ANN + Feature Attention	0.6872	0.5712	0.7863	0.6606	0.7908
	ResANN	0.6793	0.5623	0.8063	0.6607	0.78941
	ResANN + Feature Attention	0.6919	0.5777	0.7738	0.6602	0.79051
	RF-FA-ResANN	0.9029	0.8981	0.8457	0.871	0.94448
Software_defect Dataset	**SVM**	0.9588	0.7760	0.8232	0.7964	0.95239
	NAÏVE	0.7744	0.2419	0.5118	0.3124	0.73948
	Random Forest	0.9973	0.9891	0.9832	0.9860	0.99993
	ANN	0.9944	0.9921	0.9501	0.9706	0.99928
	ANN + Feature Attention	0.9981	0.9916	0.9887	0.9901	0.99997
	RANN	0.9983	0.9892	0.9939	0.9915	0.99998
	ResANN + Feature Attention	0.9983	0.9892	0.9939	0.9915	0.99998
	RF-FA-ResANN	0.9983	0.9892	0.9939	0.9915	0.99998

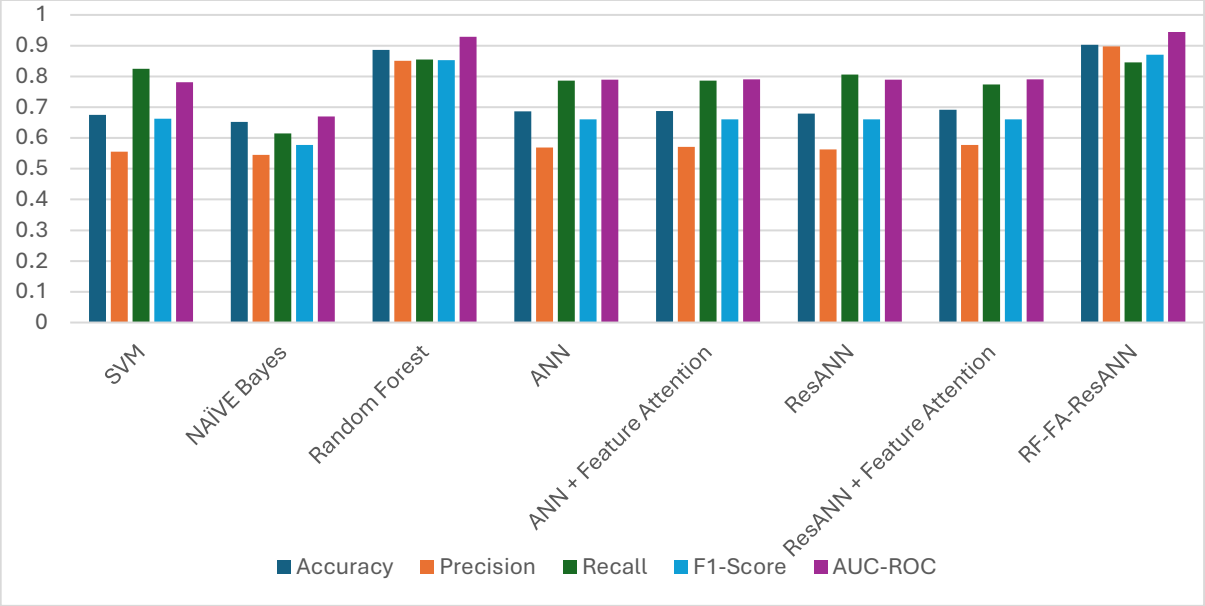


Figure 13. Comparative performance analysis of all models for NASA JM1 dataset

Figure 13 shows the comparative performance of the various baseline models with proposed RF + FA + ResANN framework for NASA JM1 dataset. The proposed model has outperformed for this dataset.

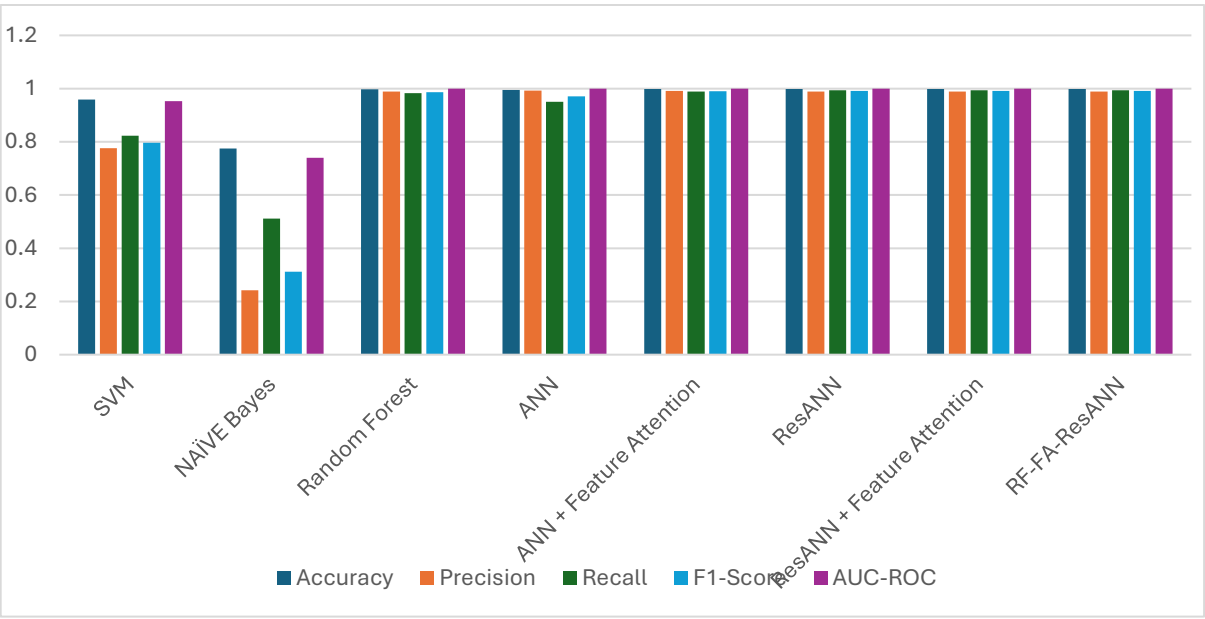


Figure 14. Comparative performance analysis of all models for Software_defect dataset

Figure 14 shows the comparative performance of the various baseline models with proposed RF + FA + ResANN framework for Software_defect dataset.

5.10 Discussion of Results

The experimental results show that the proposed RF-FA-ResANN framework consistently outperforms or achieves a high level of performance over both datasets. The improvement is more noticeable on the difficult NASA JM1 dataset, where the hybrid architecture clearly outperforms both the traditional ML and standalone deep learning architectures. The proposed model is able to provide a strong and generalized prediction model and has comparable accuracy to the most advanced models on the Software_defect dataset, which is where most models already have near-

perfect accuracy. The results support the effectiveness of the proposed combination of the Random Forest probability-based feature augmentation, Feature Attention, Residual learning, Bayesian Optimization, and SMOTE.

6. Novelty and Research Contributions

The novelty of the proposed model is the combination of Random Forest, Feature Attention, and Residual Artificial Neural Network (ResANN). The objective of proposed work is to introduce a unified framework for software defect prediction. The proposed model improves the detection of minority class with SMOTE as compared to traditional methods. Feature Attention mechanism is used to automatically highlight the most informative features. The prediction probabilities from the Random Forest are added as extra features to boost the learning process. In addition, residual connections help to prevent gradient degradation and achieve stable and efficient deep feature learning. The framework is further strengthened through Bayesian Optimization for automatic hyperparameter tuning and, deterministic training for reproducibility. Stratified 5-fold Cross-Validation with an F1-score-based optimal threshold for robust evaluation is made in this work.

Table 5. Novelty of the proposed hybrid model

Research Component	Existing Approaches	Limitation of Existing Methods	Proposed Novelty	Expected Improvement
Data Imbalance Handling	Traditional sampling or no balancing	Biased learning toward majority class	SMOTE-based oversampling before model training	Better minority class detection and improved recall
Feature Learning	Manual feature selection or direct feature input	Equal importance assigned to all features	Feature Attention Mechanism automatically learns feature importance	Improved representation of informative features
Machine Learning Model	Random Forest or ANN used independently	Limited capability to capture both global and nonlinear patterns	Hybrid Random Forest + Residual ANN architecture	Combines ensemble learning with deep feature extraction
Feature Enhancement	Original features only	Important class probability information ignored	Random Forest probability appended as an additional feature	Richer feature representation for ANN learning
Deep Learning Architecture	Conventional ANN	Gradient degradation and shallow learning	Residual Skip Connection within ANN	Better gradient propagation and deeper feature learning
Hyperparameter Optimization	Manual tuning or Grid Search	Time-consuming and may converge to suboptimal solutions	Bayesian Optimization using Keras Tuner	Faster convergence and optimal model configuration
Training Stability	Random initialization	Non-reproducible experimental results	Deterministic training with fixed random seed	Reproducible and reliable experimental outcomes
Model Evaluation	Single train-test split	Performance varies depending on data split	Stratified 5-Fold Cross-Validation	More robust and generalized evaluation
Decision Threshold	Fixed threshold (0.5)	Poor balance between precision and recall	Optimal threshold selected using Precision–Recall curve	Higher F1-score and balanced classification
Performance Assessment	Limited evaluation metrics	Incomplete assessment of classifier performance	Accuracy, Precision, Recall, F1-score, ROC-AUC with statistical analysis	Comprehensive evaluation of model effectiveness

6.1 Key Research Contributions

Table 6 summarizes the key contributions of the proposed hybrid software defect prediction model.

Table 6. Key Research Contributions

Contribution No.	Novel Contribution
C1	Integration of Random Forest and Residual ANN into a unified hybrid framework for software defect prediction.
C2	Use of Random Forest probability scores as augmented input features, enabling stacked learning without conventional ensemble voting.
C3	Introduction of a Feature Attention Module to dynamically assign importance to software metrics before classification.
C4	Incorporation of Residual Skip Connections in the ANN to improve feature propagation and reduce information loss.
C5	Automatic hyperparameter tuning using Bayesian Optimization, reducing computational cost while improving model performance.
C6	Application of SMOTE to effectively address class imbalance before hybrid model training.
C7	Deterministic implementation ensuring complete reproducibility of experimental results.
C8	Adaptive decision threshold optimization using the Precision–Recall curve instead of a fixed classification threshold.
C9	Robust validation using Stratified 5-Fold Cross-Validation and comprehensive evaluation through Accuracy, Precision, Recall, F1-score, ROC-AUC and statistical analysis.

7. Conclusion and Future Scope

Experimental results demonstrate that the proposed model outperformed conventional ML models as well as DL models. These findings confirm that the integration of ensemble learning, feature attention, residual learning, and Bayesian Optimization provides an effective and reliable framework for software defect prediction. Future work may replace Random Forest with advanced ensemble methods. These methods could be XGBoost, LightGBM, or CatBoost, that could play significant role in improving prediction performance. Moreover, future work may integrate Transformer-based attention mechanisms. Such mechanism supports better feature representation. Upcoming research might use explainable AI (XAI) techniques such as SHAP or LIME to improve model interpretability.

Acknowledgement

We thank all those who helped with the conduct of this research in any way. We are also thankful to our Institution, Department, and Laboratory for providing the necessary facilities and support.

References:

1. Jesus, M. J., Hoffmann, F., Navascués, L. J., & Sanchez, L. (2004). Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. *IEEE Transactions on Fuzzy Systems, 12*(3), 296–308. doi: 10.1109/TFUZZ.2004.825980
2. Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software, 81*(2), 186–195. doi: 10.1016/j.jss.2007.05.035
3. Immaculate, S. D., Begam, M. F., & Floramary, M. (2019). Software bug prediction using supervised machine learning algorithms. In *2019 International Conference on Data Science and Communication (IconDSC)* (pp. 1–7). IEEE. doi: 10.1109/IconDSC.2019.8816965
4. Daoud, M. S., Aftab, S., Ahmad, M., Khan, M. A., Iqbal, A., Abbas, S., Iqbal, M., & Ihnaini, B. (2022). Machine learning empowered software defect prediction system. *Intelligent Automation & Soft Computing, 31*(2*),* 1287–1300. doi: 10.32604/iasc.2022.019906
5. Rath, S. K., Sahu, M., Das, S. P., Bisoy, S. K., & Sain, M. (2022). A comparative analysis of SVM and ELM classification on software reliability prediction model. *Electronics, 11*(17), 2707. doi: 10.3390/electronics11172707
6. Mustaqeem, M., & Saqib, M. (2021). Principal component-based support vector machine (PC-SVM): A hybrid technique for software defect detection. *Cluster Computing, 24*(3), 2581–2595. doi: 10.1007/s10586-021-03282-8
7. Rahim, A., Hayat, Z., Rahim, A., Rahim, M. A., & Abbas, M. (2021). Software defect prediction with naïve Bayes classifier. In *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)* (pp. 293–297). IEEE. doi: 10.1109/IBCAST51254.2021.9393250

8. Moon, D., Lee, I., & Yoon, M. (2022). Compact feature hashing for machine learning-based malware detection. *ICT Express, 8*(1)*,* 124–129. doi: 10.1016/j.icte.2021.08.005
9. Liu, Y., Wang, Y., & Zhang, J. (2012). New machine learning algorithm: Randomforest. *In Y. Zhang (Ed.), Information Computing and Applications: Third International Conference, ICICA 2012, Chengde, China, September 14–16, 2012, Proceedings, Part III* (pp. 246–252). Springer. doi: 10.1007/978-3-642-34062-8_32
10. Nevendra, M., & Singh, P. (2021). Software defect prediction using deep learning. *Acta Polytechnica Hungarica, 18*(10), 173–189. doi: 10.12700/APH.18.10.2021.10.9
11. Khan, M. A., Elmitwally, N. S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M., & Khan, F. (2022). Software defect prediction using artificial neural networks: A systematic literature review. *Scientific Programming*, *2022*, Article 2117339. doi: 10.1155/2022/2117339
12. Munir, H. S., Ren, S., Mustafa, M., Siddique, C. N., & Qayyum, S. (2021). Attention based GRU-LSTM for software defect prediction. *PLoS ONE, 16*(3), e0247444. doi: 10.1371/journal.pone.0247444
13. Dipa, W. A., & Sunindyo, W. D. (2021, November). Software defect prediction using smote and artificial neural network. *In 2021 international conference on data and software engineering (ICoDSE)* (pp. 1-4). IEEE. doi: 10.1109/ICoDSE53690.2021.9648490
14. Bhandari, K., Kumar, K., & Sangal, A. L. (2023). Data Quality Issues in Software Fault Prediction: A Systematic Literature Review. *Artificial Intelligence Review, 56*, 7839–7908. doi: 10.1007/s10462-022-10371-6
15. Hernández-Molinos, M. J., Sánchez-García, Á. J., Barrientos-Martínez, R. E., Pérez-Arriaga, J. C., & Ocharán-Hernández, J. O. (2023). Software defect prediction with Bayesian approaches. *Mathematics, 11*(11), 2524. doi: 10.3390/math11112524
16. Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., & Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology, 17*(1), 26–40. doi: 10.11989/JEST.1674-862X.80904120
17. Singh, R., & Kumar, K. (2026). Software Fault Prediction in Service-Oriented Architecture Using Deep Learning Enabled Adaptive Optimization Technique. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering,* 1-22. doi: 10.1007/s40998-026-01019-0
18. Pandey, S., & Kumar, K. (2023). Software fault prediction for imbalanced data: A survey on recent developments. *Procedia Computer Science, 218*, 1815–1824. doi: 10.1016/j.procs.2023.01.159
19. Gong, Y., Liu, G., Xue, Y., Li, R., & Meng, L. (2023). A survey on dataset quality in machine learning. *Information and Software Technology, 162*, 107268. doi: 10.1016/j.infsof.2023.107268
20. Karthik, S., Kaliraj, S., & Lydia, M. D. (2026). Cascade framework for software fault prediction using ABC-based feature selection and SMOTE. *Discover Artificial Intelligence, 6*, 393. doi: 10.1007/s44163-026-01125-2
21. Chakraborty, A. K., & Karmakar, B. (2023, May). Software defect prediction through a hybrid approach comprising of a statistical tool and a machine learning model. In *Applications of Operational Research in Business and Industries: Proceedings of 54th Annual Conference of ORSI* (pp. 1-19). Springer Nature Singapore. doi: 10.1007/978-981-99-3247-8_1
22. Kumar, P. (2025). Machine learning based hybrid approach for software defect prediction. *International Journal of Advanced Research in Computer Science, 16*(1). doi: 10.26483/ijarcs.v16i1.7192
23. Siddika, A., Begum, M., Al Farid, F., Uddin, J., & Karim, H. A. (2025). Enhancing software defect prediction using ensemble techniques and diverse machine learning paradigms. *Eng, 6*(7), 161. doi: 10.3390/eng6070161
24. Tamrakar, P. K., Roy, D., Agarwal, P., Ghemas, M. F., Ghosh, S. M., KS, R., & Mohil, M. (2026). Innovative approaches to software defect prediction using ensemble learning models. *Future Technology, 5*(1), 38-46. doi: 10.55670/fpll.futech.5.1.4
25. Matloob, F., Ghazal, T. M., Taleb, N., Aftab, S., Ahmad, M., Khan, M. A., Abbas, S., & Soomro, T. R. (2021). Software defect prediction using ensemble learning: A systematic literature review. *IEEE Access, 9*, 98754–98771. doi: 10.1109/ACCESS.2021.3095559
26. Sharma, T., Jatain, A., Bhaskar, S., & Pabreja, K. (2023). Ensemble machine learning paradigms in software defect prediction. *Procedia Computer Science, 218*, 199–209. doi: 10.1016/j.procs.2023.01.002
27. Son, L. H., Pritam, N., Khari, M., Kumar, R., Phuong, P. T. M., & Thong, P. H. (2019). Empirical Study of Software Defect Prediction: A Systematic Mapping. *Symmetry, 11*(2), 212. doi: 10.3390/sym11020212
28. Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2024). Machine learning based methods for software fault prediction: A survey. *Journal of Systems and Software, 218*, Article 112175. doi: 10.1016/j.jss.2024.112175