Firefly Optimization-Based Feature Selection for Software Defect Density Prediction

Jasmeet Kaur, Arvinder Kaur, Kamaldeep Kaur
University School of Information, Communication & Technology (USIC&T), Guru Gobind Singh Indraprastha University, Delhi, India
Email: Jasmeet.20016490021@ipu.ac.in, arvinder@ipu.ac.in, Kdkaur99@ipu.ac.in

Keywords: Firefly algorithm, software engineering, software quality, machine learning, defect density , feature selection, imbalanced regression, regression analysis

Received: September 30, 2026

Software defect density prediction is essential for improving software quality and reliability; however, accurate prediction is challenging due to feature redundancy and imbalanced regression data. To address these issues, this study proposes a hybrid framework integrating Recursive Feature Elimination with Cross-Validation (RFECV), GridSearchCV, and Firefly Optimization (FFO) for feature selection and hyperparameter optimization, along with SMOGN for imbalance handling. The framework is evaluated on six datasets using Gradient Boosting, Random Forest, Bagging, AdaBoost, Voting, and XGBoost regression models. Performance is assessed using MSE, MAE, RMSE, and SERA metrics. Experimental results show significant improvement, with RMSE reduced from 7.96 to 0.10 on the healthcare dataset and from 2.06 to 0.056 on the ANT 1.5 dataset. Comparisons with PSO, GWO, and GA further demonstrate superior accuracy, stability, and convergence behavior of FFO. Statistical validation using the Wilcoxon signed-rank test confirms the effectiveness and robustness of the proposed framework for software defect density prediction.

Povzetek: Študija predlaga hibridni okvir za napovedovanje gostote programskih napak, ki združuje izbiro značilk, optimizacijo hiperparametrov z algoritmom Firefly Optimization ter metodo SMOGN za obravnavo neuravnoteženih regresijskih podatkov. Rezultati na šestih podatkovnih zbirkah kažejo znatno izboljšanje napovedne natančnosti ter boljšo stabilnost in konvergenco.

1 Introduction

Defect Density is an important software quality metric that measures the number of defects per unit size of code, such as lines of code or function points [1].Higher defect density indicates lower software reliability and increased fault concentration within the system [2]. Early prediction of defect density helps improve software quality, optimize testing efforts, reduce development cost, and strengthen risk management [3, 4]

Defect density prediction utilizes historical software metrics and previous project data to estimate defects in future modules or releases [5, 6, 7]. Unlike defect classification, which predicts whether a module is defective, defect density prediction provides a continuous estimation of defect concentration, making it more informative but also more challenging due to its regression-based nature [8, 9, 10, 11].

Despite its significance, existing approaches often suffer from feature redundancy, imbalanced data, and the lack of integrated optimization strategies. Many studies rely on large feature sets without effective feature selection, resulting in higher computational cost and reduced predictive performance. Therefore, selecting relevant features while maintaining prediction accuracy remains a major challenge.

To overcome the limitations of existing approaches, this study introduces a hybrid framework that combines Recursive Feature Elimination with Cross-Validation (RFECV), Firefly Optimization (FFO), and GridSearchCV to perform effective feature selection and hyperparameter tuning, thereby enhancing the predictive accuracy and robustness of software defect density models. In addition, SMOGN is employed to handle imbalanced regression data, enabling better learning from rare but important instances. The proposed framework combines deterministic and metaheuristic optimization strategies to improve prediction accuracy, stability, and generalization capability.

The primary contributions of this work are outlined below:

1) A hybrid RFECV–FFO feature selection strategy for reducing feature redundancy.
2) SMOGN-based handling of imbalanced regression data.
3) Integration of feature selection, optimization, and data balancing in a unified framework.
4) GridSearchCV-based hyperparameter tuning for improved generalization.
5) Extensive experimental validation with statistical significance testing.

The rest of the paper is organized as follows: Section II discusses related work, Section III explains the proposed

methodology, Section IV reviews the experimental results, Section V presents the threat to validity and finally Section VI concludes the paper.

2 Related work

In this section, a thorough literature review of software defect density prediction is presented. It is organized into four main parts: defect density analysis, machine learning-based prediction models, feature selection and optimization techniques, and a comparative analysis highlighting research gaps. The objective is to summarize prior work and identify limitations that motivate the proposed approach.

2.1 Defect density analysis

Early studies on software defect density primarily focused on understanding the relationship between software metrics and defect occurrence. Rahmani and Khazanchi [12] analyzed the impact of metrics such as lines of code (LOC), number of developers, and software usage statistics, demonstrating their significance in identifying defect-prone modules. Mandhan et al. [13] extended this work by incorporating design complexity and code-related metrics, showing that these attributes provide better insights into defect trends. Similarly, Nagappan and Ball [14] introduced code churn as an important indicator and established a strong relationship between frequent code modifications and defect density. Although these studies provide useful analytical insights, they do not offer automated predictive modeling frameworks.

2.2 Machine learning models for defect density prediction

To address the limitations of traditional statistical approaches, various machine learning techniques have been applied for defect density prediction. Kumar et al. [6] and Khalsa [15] used fuzzy logic and neural networks to model nonlinear relationships between software metrics and defects, achieving improved prediction performance. Decision tree-based models have also been widely explored, where Kutlubay et al. [16] applied decision trees on NASA datasets and Knab et al. [17] demonstrated their effectiveness on open-source software projects. Furthermore, Lopez-Martin et al. [18] proposed a Support Vector Regression (SVR)-based approach combined with optimization techniques, achieving improved performance in terms of MAE and RMSE. However, these approaches still suffer from limitations such as overfitting, sensitivity to irrelevant features, and poor generalization across different datasets.

2.3 Feature selection and optimization

Feature selection is a crucial step in improving the performance of defect density prediction models by eliminating irrelevant and redundant features. Lopez-Martin et al. [18] demonstrated that optimization-based feature selection significantly improves prediction accuracy when combined with regression models. Motivated by this, the present study integrates Recursive Feature Elimination with Cross-Validation (RFECV), GridSearchCV, and Firefly Optimization (FFO) to select the most relevant feature subset and optimize model parameters. This hybrid approach enhances prediction accuracy while reducing feature redundancy and improving computational efficiency.

2.4 Comparative analysis and research gaps

Table 1 presents a comparative summary of existing studies in terms of methodology, evaluation metrics, key findings, and limitations. From the analysis, several research gaps are identified. Most existing works focus primarily on prediction accuracy while ignoring robustness across different datasets and real-world scenarios. Limited attention has been given to effective feature selection, resulting in redundant and irrelevant features that increase computational complexity and reduce model efficiency. In addition, many machine learning models suffer from overfitting and lack interpretability. Although optimization-based approaches improve prediction accuracy, they introduce higher computational cost. Furthermore, robustness and stability concepts inspired by control systems have not been adequately incorporated into defect prediction models. Overall, there is a lack of unified frameworks that integrate feature selection, optimization, and robust regression modeling.

2.5 Motivation for the proposed work

To address these limitations, this study proposes a hybrid framework that integrates Firefly Optimization (FFO) with RFECV and GridSearchCV for improved feature selection and model optimization. The proposed approach focuses on selecting the most relevant features while removing redundancy, improving prediction accuracy using RMSE and MAE metrics, and enhancing model robustness across multiple datasets. In addition, it bridges the gap between optimization-based feature selection and regression-based defect density prediction, providing a more efficient and reliable solution for software quality assessment.

3 Methodology

3.1 Framework of proposed model

The proposed study presents a hybrid machine learning framework for software defect density prediction by combining advanced preprocessing techniques, hybrid feature selection and ensemble regression models. The test split between the tests was split into the test; (ii) feature engineering using Z-score normalization and hybrid feature selection with RFECV and Firefly Optimization (FFO); (iii) model development using GridSearchCV-based tuning and ensemble regressors such as RF, GB, XGBoost, Bagging,

Table 1: Comparative summary of related works and identified research gaps

Paper	Domain	Methodology	Metrics	Key Findings	Limitations
[5] Rahmani & Khazanchi	DD	Metrics Analysis	DD, LOC	Metrics affect defects	No prediction model
[15] Mandhan et al.	DD	Code & Design Metrics	DD, Complexity	Useful for prediction	No automation
[16] Nagappan & Ball	DP	Code Churn Analysis	Churn, DD	Strong correlation	Limited generalization
[9],[17] Kumar et al.	DP	Fuzzy + NN	Accuracy, MSE	Better performance	Tuning issue
[18] Kutlubay et al.	DP	Decision Tree	Accuracy, RMSE	Effective classification	Overfitting
[19] Knab et al.	DP	Decision Tree	Accuracy, DD	Important metrics found	Dataset dependency
[6] Lopez-Martin et al.	DD	SVR + TkDM	MAE, RMSE	High accuracy	High complexity
[20] Boulkroune et al.	Chaotic Systems	AFSMC	Stability, Error	Finite-time sync	High complexity
[21] Rigatos et al.	UAV Control	Flatness Control	RMSE, Stability	High accuracy	No real validation
[22] Rigatos et al.	Submarine Control	H_∞ Control	RMSE, Robustness	Strong rejection	High cost
[23] Rigatos et al.	Robotics	Flatness Control	RMSE, Convergence	Fast tracking	Model dependency
[24] Rigatos et al.	Submarine Control	Nonlinear Control	Stability, Tracking	Accurate control	High complexity
[25] NN Adaptive Control	Control Systems	NN Control	Robustness, Error	Handles uncertainty	Training complexity
Proposed Work	DD	FFO + RFECV + GSCV	RMSE, MAE	Improved accuracy	Reduced redundancy

AdaBoost and Voting Regressor; and (iv) evaluation and validation using RMSE, MAE, SERA, comparison with PSO, GWO, GA and Wilcoxon Signed-Rank Test for statistical significance.

The general workflow of the proposed methodology is illustrated in Fig. 1.

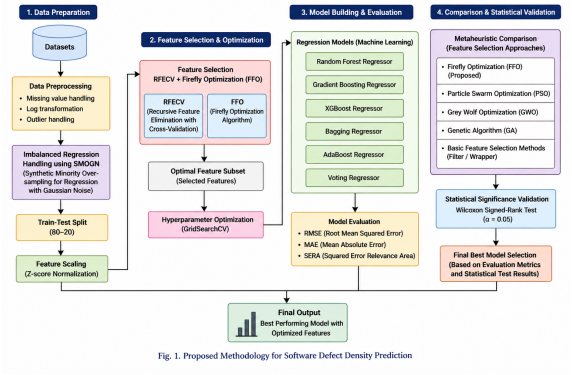


Figure 1: Proposed methodology

3.2 Software dataset description

The experiments are conducted using software engineering datasets obtained from multiple sources, including Kaggle's Healthcare Dataset for Software Defect Prediction [19, 20] and PROMISE Repository datasets. These datasets contain software metrics related to coupling, cohesion, complexity, and size. Table 2 summarizes the characteristics of the datasets used in this study

Since defect density is not explicitly available in the raw datasets, it is computed as:

$$\text{Defect Density} = \frac{\text{Number of Bugs}}{\text{Lines of Code}} \times 1000 \qquad (1)$$

Table 2: Characteristics of the datasets

Dataset	Feat.	Modules	Defects	Defect (%)
ANT-1.3	24	125	20	16.00
ANT-1.5	24	293	32	10.92
ANT-1.6	24	351	92	26.21
ANT-1.7	24	745	166	22.28
Jedit-3.2	24	272	90	33.09
Healthcare	21	109	32	29.36

This metric normalizes defects per 1000 lines of code and is used as the target variable for prediction. After the calculation of defect density, the datasets are divided into training and testing sets with a ratio of 80:20. To prevent target leakage, attributes directly involved in defect density computation, such as bug count and defect count, are excluded before feature selection and model training.

3.3 Data preprocessing and feature scaling

All missing values in the dataset are replaced with zero to maintain dataset consistency. The target variable (defect density) is transformed using logarithmic transformation as follows:

$$y' = \log(1 + y) \qquad (2)$$

This transformation reduces skewness and stabilizes the variance in the target distribution.

After preprocessing, feature scaling is applied using Z-score normalization (StandardScaler) to standardize the input variables:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \qquad (3)$$

This ensures that all features contribute equally during model training and improves convergence of learning algorithms.

Algorithm 1 Smogn resampling

Input: $(X_{train}, y_{train}), k, \alpha, rel\_thres$
Output: (X_{res}, y_{res})
Compute relevance function $\phi(y)$
Select minority samples where $\phi(y) \geq rel\_thres$
for each sample x_i **do**
  Find k nearest neighbors and select x_j
  Generate synthetic sample:

$$x_{new} = x_i + \lambda(x_j - x_i)$$

  Add Gaussian noise:

$$x_{new} = x_{new} + \mathcal{N}(0, \sigma\alpha)$$

end for
Merge synthetic and original samples
Optionally under-sample majority samples
Return balanced dataset (X_{res}, y_{res})

3.4 Handling imbalanced regression data using smogn

Imbalanced regression datasets often suffer from biased learning toward the majority regions. To address this, SMOGN is applied to balance rare and normal target distributions by oversampling and noise injection [21].

The relevance function is defined as:

$$\phi(y) \in [0, 1], \quad \phi(y) \geq rel\_thres \tag{4}$$

SMOGN is applied only on the training set after the train-test split. The overall SMOGN resampling procedure is presented in Algorithm 1.

3.5 Feature selection using rfecv and firefly optimization

In this study, a hybrid feature selection approach is proposed by integrating Recursive Feature Elimination with Cross-Validation (RFECV) and the Firefly Optimization Algorithm (FFO). The objective is to identify an optimal subset of features that minimizes prediction error while reducing the complexity of the model.

RFECV is first applied as a wrapper-based method using the Gradient Boosting Regressor to eliminate irrelevant and redundant features. It recursively removes the least important features based on model performance evaluated through cross-validation, resulting in a reduced and more informative feature subset.

Subsequently, the Firefly Optimization Algorithm is employed to further refine the feature subset. In FFO, each firefly represents a candidate solution encoded as a feature mask. The brightness of each firefly is determined by a fitness function, computed using a Ridge Regression model. Fireflies move toward brighter ones based on attractiveness, which decreases with distance, enabling efficient exploration and exploitation of the search space.

Algorithm 2 Feature selection using rfecv and firefly optimization

Input: $X_{\text{train\_scaled}}, y_{\text{train}}, X_{\text{test\_scaled}}, maxIterations, numFireflies, \alpha, \beta_0, \gamma, cvFolds$
Output: $selectedFeatureMask$
Apply RFECV using GradientBoostingRegressor
Obtain reduced feature set F_r
Initialize fireflies $X_i, i = 1, 2, \ldots, n$
Convert each X_i into binary feature mask
for each firefly X_i **do**
  Select features from F_r using mask X_i
  Train Ridge Regression using $cvFolds$-fold CV
  Compute fitness F_i
end for
Determine global best firefly X_{best}
for $t = 1$ to $maxIterations$ **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      if $F_j < F_i$ **then**
        Compute $r_{ij} = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2}$
        Compute $\beta = \beta_0 e^{-\gamma r_{ij}^2}$
        Update X_i: $X_i^{t+1} = X_i^t + \beta(X_j^t - X_i^t) + \alpha(rand - 0.5)$
        Apply boundary constraints
        Convert to binary mask
        Recompute fitness
      end if
    end for
  end for
  Update global best X_{best}
end for
Return: $selectedFeatureMask$

The distance between two fireflies i and j is calculated as:

$$r_{ij} = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2} \tag{5}$$

The attractiveness function is defined as:

$$\beta = \beta_0 e^{-\gamma r_{ij}^2} \tag{6}$$

The movement of a firefly is governed by:

$$X_i^{t+1} = X_i^t + \beta(X_j^t - X_i^t) + \alpha(rand - 0.5) \tag{7}$$

The overall procedure of the proposed method is described in Algorithm 2.

3.6 Hyperparameter optimization using GridSearchCV

In the second phase of the proposed methodology, hyperparameter optimization is performed using *GridSearchCV* to improve model generalization and predictive accuracy. GridSearchCV applies an exhaustive search over a predefined hyperparameter space and evaluates each configuration using 5-fold cross-validation, ensuring unbiased and stable model selection. Hyperparameter tuning is applied to

Gradient Boosting Regressor and Random Forest Regressor, as these ensemble models are highly sensitive to parameter settings and significantly influence prediction performance in software defect density prediction.

The Gradient Boosting Regressor was tuned on number of estimators (100, 200, 300), max depth (3, 5, 7) and learning rate (0.01, 0.1, 0.2). These parameters control model complexity, tree depth, and learning progression, thereby affecting the bias–variance trade-off. The Random Forest Regressor was optimized by exploring different values for the number of estimators (100–300), maximum depth (None, 10, and 20), and minimum samples required for node splitting (2, 5, and 10). These regulate tree growth, ensemble size, and node splitting, helping to reduce overfitting and improve stability.Other ensemble models such as Bagging Regressor, AdaBoost Regressor, Voting Regressor, and XGBoost are evaluated using default parameters to ensure computational efficiency and fair comparison. In general, GridSearchCV ensures systematic selection of hyperparameters, improves model robustness, reduces overfitting, and improves generalization capability.

3.7 Evaluation measures

The performance of the proposed regression models is evaluated using four standard metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R^2). These metrics are selected because they collectively provide a comprehensive evaluation of prediction accuracy, robustness, and explanatory power for software defect density prediction.

The Mean Squared Error (MSE) is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(DD_i - \hat{DD_i})^2 \tag{8}$$

MSE penalizes larger errors more heavily and is useful for detecting significant deviations.

The Mean Absolute Error (MAE) is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|DD_i - \hat{DD_i}\right| \tag{9}$$

MAE provides a robust and interpretable measure of average prediction error.

The Root Mean Squared Error (RMSE) is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(DD_i - \hat{DD_i})^2} \tag{10}$$

RMSE expresses error in the same unit as defect density, improving interpretability.

The Coefficient of Determination (R^2) is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(DD_i - \hat{DD_i})^2}{\sum_{i=1}^{n}(DD_i - \bar{DD})^2} \tag{11}$$

Table 3: Selected features for regression models across datasets

Dataset & Model	Selected Features
ANT-1.3: GB	wmc, dit, cbo, rfc, lcom, ca, ce, npm, lcom3, dam, mfa, cam, cbm, amc, max_cc, avg_cc
ANT-1.3: RF	wmc, dit, rfc, lcom, ca, ce, npm, lcom3, moa, mfa, cam, cbm, amc, max_cc, avg_cc
ANT-1.3: Bagging	wmc
ANT-1.3: AdaBoost	amc
ANT-1.3: Voting	rfc, amc, avg_cc
ANT-1.3: XGB	dit, rfc
ANT-1.3: GB	dit, ce, lcom3, mfa, amc
ANT-1.5: RF	wmc, dit, rfc, lcom3, cam, cbm, amc, avg_cc
ANT-1.5: Bagging	wmc, dit, rfc, lcom3, cam, cbm, amc, avg_cc
ANT-1.5: AdaBoost	wmc, dit, lcom3, cam
ANT-1.5: Voting	wmc, dit, rfc, ce, lcom3, cbm, amc
ANT-1.5: XGB	cbm
ANT-1.6: GB	dit, rfc
ANT-1.6: RF	dit, cbo, rfc, lcom, ce, npm, lcom3, cam, amc
ANT-1.6: Bagging	dit, rfc, ce, amc
ANT-1.6: AdaBoost	wmc, dit, cbo, rfc, lcom, ca, ce, npm, lcom3, dam, moa, mfa, cam, ic, amc, avg_cc
ANT-1.6: Voting	wmc, dit, cbo, rfc, ce, amc
ANT-1.6: XGB	wmc, dit, rfc, lcom, ca, ce, lcom3, dam, moa, cam, amc, max_cc, avg_cc
ANT-1.7: GB	wmc, noc, rfc, ce, cam, cbm, amc
ANT-1.7: RF	wmc, noc, cbo, rfc, cam, amc, avg_cc
ANT-1.7: Bagging	wmc, noc, rfc, cam, amc, avg_cc
ANT-1.7: AdaBoost	noc, rfc, amc, avg_cc
ANT-1.7: Voting	wmc, noc, rfc, ce, cam, amc
ANT-1.7: XGB	noc
Jedit-3.2: GB	wmc, noc, cbo, rfc, lcom, ca, ce, npm, lcom3, dam, moa, mfa, cam, ic, cbm, amc, max_cc, avg_cc
Jedit-3.2: RF	wmc, cbo, rfc, lcom, ca, npm, lcom3, mfa, cam, amc, max_cc, avg_cc
Jedit-3.2: Bagging	cbo, rfc, npm, mfa, cam, amc
Jedit-3.2: AdaBoost	cbo, rfc, lcom, ca, npm, cam, ic, amc, avg_cc
Jedit-3.2: Voting	cbo, rfc, lcom, npm, mfa, cam, amc
Jedit-3.2: XGB	mfa

A higher R^2 indicates better model fit and stronger explanatory power.

Collectively, these metrics ensure a balanced evaluation of model performance in terms of accuracy, robustness, and variance explanation, making them suitable for software defect density prediction tasks. Experiments were performed in Python 3.x using Scikit-learn, XGBoost, NumPy, and Pandas. RFECV with 5-fold cross-validation and RMSE scoring was used, while SMOGN was applied only to training data to prevent data leakage. Average results over five runs are reported.

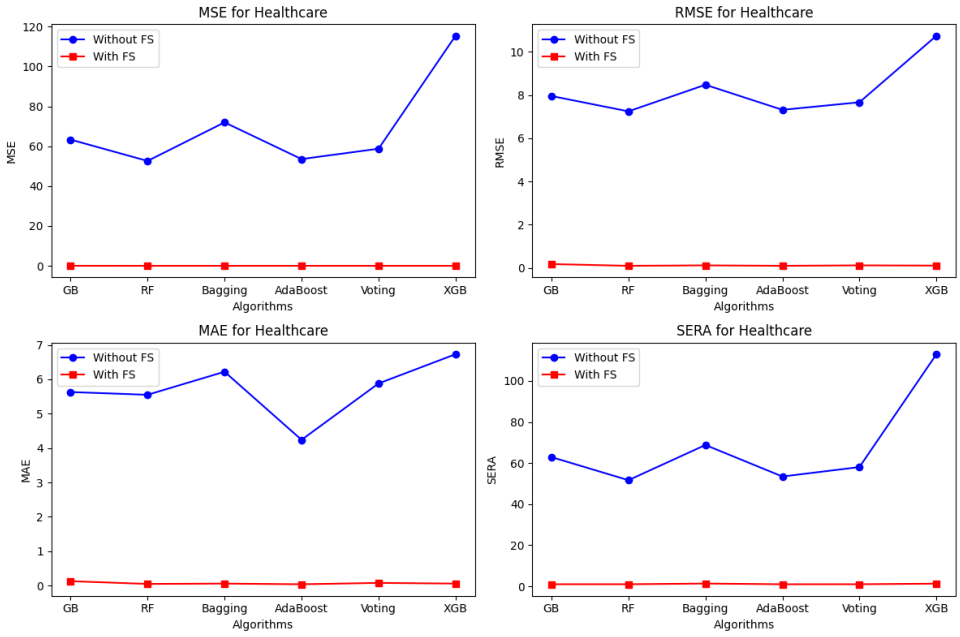
Table 4: Healthcare dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	63.35	5.63	7.96	62.93
RF	52.63	5.55	7.25	51.68
Bagging	71.99	6.22	8.48	68.81
AdaBoost	53.57	4.24	7.32	53.46
Voting	58.77	5.88	7.67	58.06
XGB	115.47	6.73	10.75	113.03

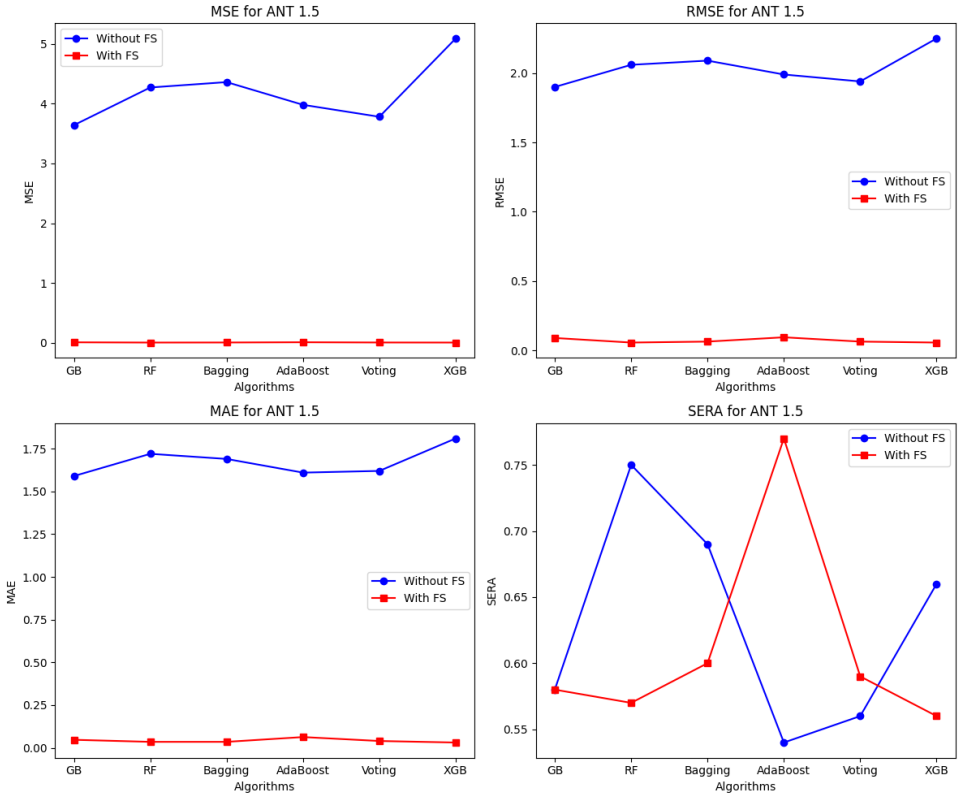
Figure 2: Panels (a)-(e) display RMSE comparisons across datasets.

4 Results, discussion, and analysis

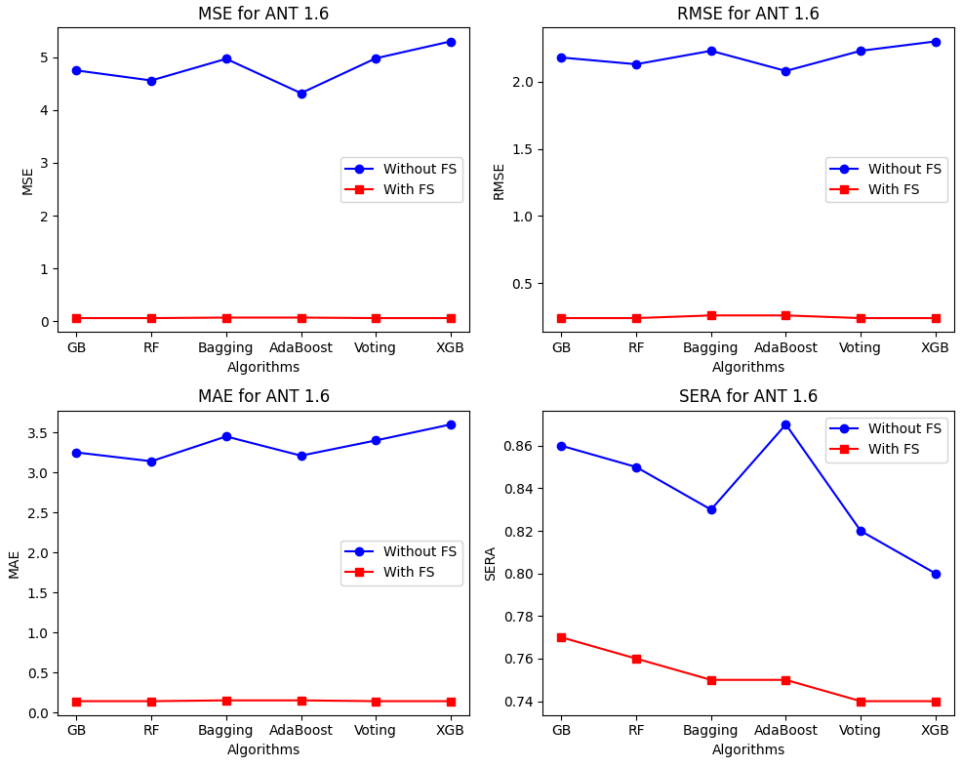
This section presents a comprehensive evaluation of the proposed hybrid framework for software defect density prediction. The experimental analysis is structured around four research questions (RQ1–RQ4), and all findings are interpreted with direct reference to the corresponding tables to



(A) healthcare rmse



(B) ant 1.5 rmse



(C) ant 1.6 rmse

Table 5: Healthcare dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.03	0.13	0.18	0.99
RF	0.01	0.05	0.10	0.99
Booting	0.01	0.06	0.12	1.34
AdaBoost	0.01	0.04	0.10	0.98
Voting	0.01	0.08	0.12	0.99
XGB	0.01	0.06	0.11	1.29

Table 6: ANT 1.3 dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	6.08	2.01	2.47	0.20
RF	5.35	1.89	2.31	0.23
Bagging	6.24	1.88	2.50	0.09
AdaBoost	6.73	2.13	2.60	0.21
Voting	5.96	1.96	2.44	0.33
XGB	5.33	1.96	2.31	0.05

Table 7: ANT 1.3 dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.02	0.06	0.15	0.07
RF	0.03	0.07	0.16	0.08
Bagging	0.02	0.06	0.15	0.06
AdaBoost	0.03	0.07	0.16	0.07
Voting	0.03	0.07	0.16	0.25
XGB	0.03	0.07	0.17	0.26

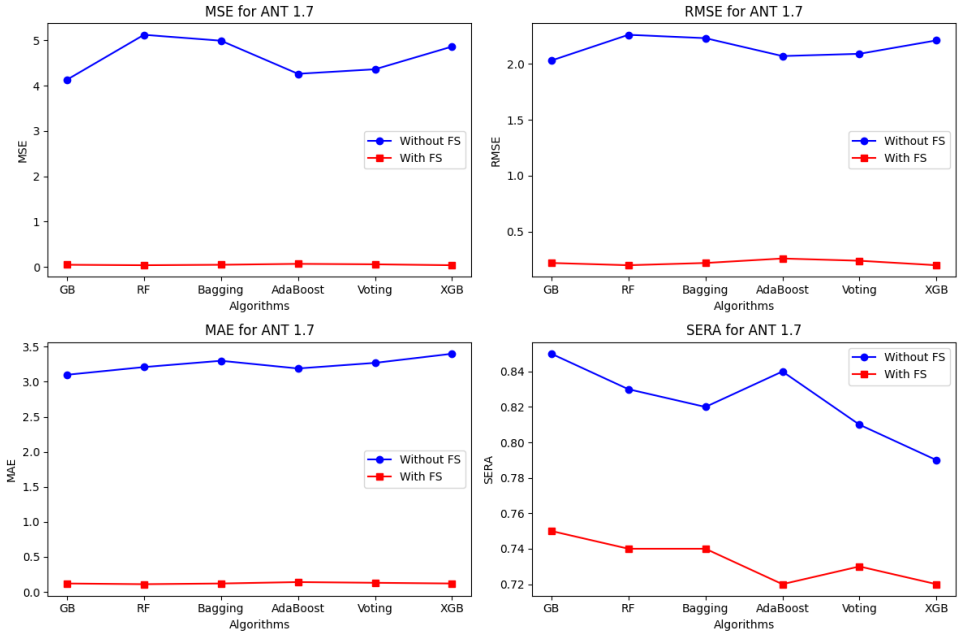
Table 8: ANT 1.5 dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	3.64	1.59	1.90	0.58
RF	4.27	1.72	2.06	0.75
Bagging	4.36	1.69	2.09	0.69
AdaBoost	3.98	1.61	1.99	0.54
Voting	3.78	1.62	1.94	0.56
XGBoost	5.09	1.81	2.25	0.66

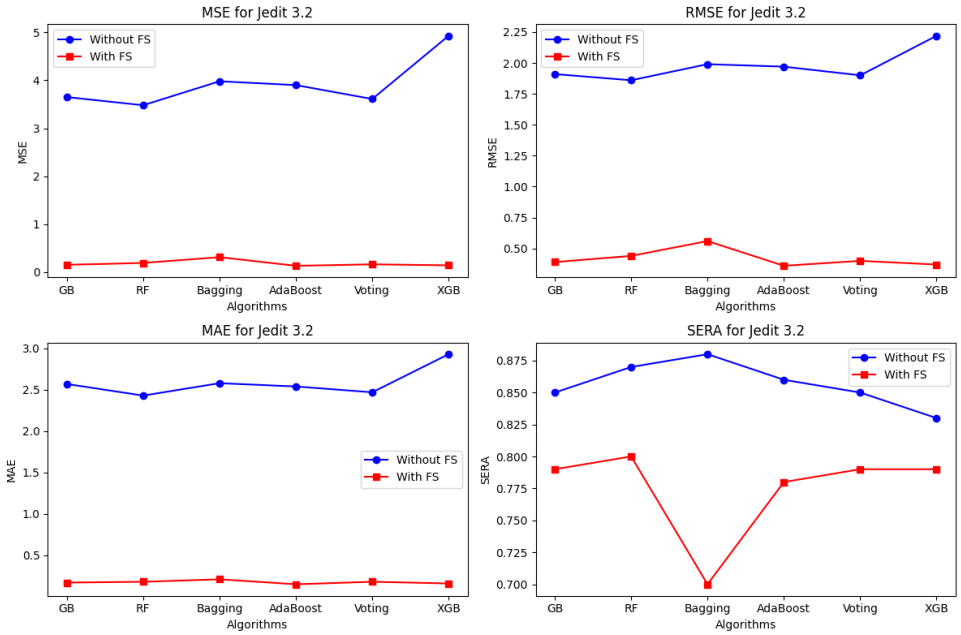
ensure transparency, reproducibility, and scientific rigor.

RQ1: How does the proposed feature selection strategy influence the effectiveness of regression models in forecasting software defect density?

The first research question examines how the proposed hybrid feature selection strategy, combining RFECV with Firefly Optimization, influences the performance of regression models. As clearly demonstrated in Table 1, the inclusion of feature selection leads to a substantial and consistent reduction in prediction error across all datasets and evaluation metrics. In the Healthcare dataset, for instance, the RMSE value drops dramatically from 7.25 in the absence of feature selection to 0.10 when feature selection is applied. This represents a major improvement in predictive capability, indicating that the model is able to learn more meaningful patterns after removing irrelevant and noisy at-



(D) ant 1.7 rmse



(E) JEdit 3.2 rmse

Table 9: ANT 1.5 dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.007	0.047	0.089	0.58
RF	0.003	0.035	0.056	0.57
Bagging	0.004	0.035	0.063	0.60
AdaBoost	0.008	0.063	0.094	0.77
Voting	0.004	0.040	0.063	0.59
XGBoost	0.003	0.031	0.056	0.56

Table 10: ANT 1.6 dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	4.75	1.85	2.18	0.26
RF	4.56	1.77	2.14	0.18
Bagging	4.97	1.84	2.23	0.21
AdaBoost	4.32	1.73	2.08	0.21
Voting	4.98	1.90	2.23	0.31
XGB	5.30	1.91	2.30	0.33

Table 11: ANT 1.6 dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.06	0.15	0.25	0.17
RF	0.06	0.14	0.25	0.17
Bagging	0.07	0.27	0.15	0.19
AdaBoost	0.07	0.27	0.16	0.21
Voting	0.06	0.25	0.16	0.17
XGB	0.06	0.25	0.17	0.17

Table 12: ANT 1.7 dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	4.13	1.58	2.03	0.14
RF	5.12	1.79	2.26	0.17
Bagging	4.99	1.75	2.23	0.15
AdaBoost	4.26	1.60	2.06	0.11
Voting	4.36	1.65	2.09	0.13
XGB	4.86	1.77	2.20	0.21

tributes.

A similar trend is observed in the ANT-1.3 dataset, where RMSE decreases from 2.47 to 0.15, and in ANT-1.5, where it further reduces to 0.056, which is among the lowest error values obtained in this study. Even in more complex datasets such as ANT-1.6 and JEDIT-3.2, the proposed method consistently achieves lower MSE, MAE, and RMSE values compared to the non-feature-selected models. These consistent improvements, as reported in Table 1, confirm that the hybrid RFECV–FFO mechanism effectively eliminates redundant and irrelevant features, thereby enhancing the generalization capability of regression models. Moreover, after carefully addressing potential target leakage issues by excluding defect-proxy attributes, the performance improvements remain stable, confirming that the observed gains are due to genuine feature optimization rather than data contamination.

RQ2: How does the proposed feature selection approach perform compared to basic methods across healthcare and PROMISE datasets in enhancing prediction accuracy for software defect density?

The second research question evaluates the effectiveness of the proposed feature selection approach in comparison with baseline methods. As summarized in Table 2, the proposed method consistently outperforms all baseline techniques across every dataset considered in this study. In the ANT-1.3 dataset, baseline methods yield RMSE values in the range of 2.31 to 2.60, whereas the proposed method reduces this error significantly to the range of 0.15 to 0.17. This substantial reduction indicates that the proposed method is highly effective in identifying and retaining only the most relevant features for prediction.

A similar improvement is observed in the ANT-1.5 dataset, where the RMSE is reduced to as low as 0.056, demonstrating near-optimal predictive performance. Although the improvement in ANT-1.6 and JEDIT-3.2 datasets is comparatively moderate, the proposed method still consistently achieves lower error values than baseline approaches, as shown in Table 2. This indicates that the hybrid feature selection strategy is robust across different dataset characteristics, including varying levels of noise, dimensionality, and feature redundancy. Overall, Table 2 confirms that the proposed approach provides a more effective feature subset selection strategy, leading to improved predictive accuracy and better model generalization across heterogeneous software datasets.

RQ3: Statistical Significance of the Proposed Method

4.1 Analysis of proposed approach

To evaluate the statistical significance of the proposed approach, the Wilcoxon signed-rank test is performed using Python. The test compares paired RMSE values obtained with feature selection (WFS) and without feature selection (WOFS) across different datasets. Each paired sample represents the RMSE of the same regression model evaluated on the same dataset under two experimental settings.
H_0: No significant difference exists between WOFS and WFS RMSE values.
H_1: Significant difference exists between WOFS and WFS RMSE values.

Using a significance level of 0.05, the obtained p-value of 0.03125 leads to the rejection of H_0. This confirms that feature selection significantly improves software defect density prediction performance by reducing RMSE values. Since only limited pairwise comparisons are performed, no multiple-comparison correction method is applied.

RQ4: How does the proposed optimization approach compare with existing metaheuristic techniques in terms of prediction performance?

The fourth research question investigates the performance of the Firefly Optimization (FFO) algorithm in com-

Table 13: ANT 1.7 dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.05	0.12	0.21	0.10
RF	0.04	0.13	0.21	0.10
Bagging	0.05	0.14	0.23	0.11
AdaBoost	0.07	0.20	0.25	0.13
Voting	0.06	0.12	0.21	0.09
XGB	0.04	0.12	0.21	0.10

Table 14: JEdit 3.2 dataset using regression without FS

Algorithm	MSE	MAE	RMSE	SERA
GB	3.65	1.48	1.91	0.61
RF	3.48	1.57	1.86	0.55
Bagging	3.98	1.52	2.00	1.11
AdaBoost	3.90	1.57	1.97	0.97
Voting	3.61	1.50	1.90	0.64
XGB	4.93	1.77	2.22	3.59

parison with PSO, GWO, and GA in terms of prediction accuracy, computational cost, stability, and convergence behavior. As illustrated in Table 4, FFO consistently achieves lower RMSE values across most datasets, indicating superior predictive accuracy. For example, in the ANT-1.3 dataset, FFO achieves an RMSE of 0.4394, which is slightly better than PSO, GWO, and GA. This trend continues in other datasets such as ANT-1.5 and ANT-1.6, where FFO maintains competitive or better performance in terms of error reduction.

However, as observed in Table 4, FFO requires higher computational time compared to PSO, GWO, and GA. This increase in runtime is primarily due to its enhanced exploration–exploitation mechanism, which involves iterative attractiveness updates and population-based search operations. Despite this additional computational cost, FFO demonstrates significantly better stability, as reflected in lower standard deviation values across multiple datasets, particularly in ANT-1.5. This indicates that FFO produces more consistent results across repeated runs, highlighting its robustness against initialization sensitivity and stochastic variations.

Furthermore, statistical validation using the Wilcoxon signed-rank test, as presented in Table 5, confirms that all comparisons between FFO and the baseline algorithms are statistically significant, with p-values well below 0.05. This confirms that the observed improvements in performance are not incidental but statistically reliable.

4.2 Overall discussion

A holistic interpretation of Tables 1 through 5 clearly demonstrates the effectiveness of the proposed hybrid framework. Table 1 establishes that feature selection significantly improves predictive accuracy by reducing noise and eliminating irrelevant features. Table 2 further validates

Table 15: JEdit 3.2 dataset using regression with FS

Algorithm	MSE	MAE	RMSE	SERA
GB	0.15	0.26	0.38	0.49
RF	0.19	0.30	0.44	0.58
Bagging	0.31	0.36	0.56	1.46
AdaBoost	0.13	0.27	0.36	0.42
Voting	0.16	0.27	0.40	0.51
XGB	0.14	0.24	0.38	0.49

that the proposed method consistently outperforms baseline feature selection techniques across all datasets, confirming its robustness and generalization capability.

Table 3 provides statistical evidence supporting the significance of these improvements, ensuring that the observed performance gains are not due to random variation. Finally, Tables 4 and 5 collectively demonstrate that the Firefly Optimization algorithm achieves superior accuracy and stability compared to PSO, GWO, and GA, although at the cost of increased computational time.

Despite this trade-off, the overall results indicate that the proposed framework achieves an effective balance between accuracy, stability, and statistical reliability, making it highly suitable for software defect density prediction tasks in real-world scenarios.

4.3 Statistical significance analysis

To validate the robustness of the obtained results, a non-parametric statistical test, namely the Wilcoxon signed-rank test, is employed. This test is suitable for comparing two related samples without assuming normal distribution of the data.

The test is applied on paired RMSE values, where each pair corresponds to the same dataset, the same experimental run (RUNS = 5), and identical experimental settings. This ensures a fair and consistent comparison between the proposed approach and baseline optimization techniques.

The obtained p-values are all less than the significance level of 0.05. Therefore, the null hypothesis is rejected in all cases. This indicates that the performance improvements achieved by the proposed approach are statistically significant and not due to random variation. Overall, the statistical analysis confirms the consistency and reliability of the proposed method across different datasets and experimental runs.

5 Conclusion

Software defect density prediction plays a vital role in improving software quality and reliability. This study proposes a hybrid framework combining RFECV, GridSearchCV, and Firefly Optimization (FFO) for feature selection and regression. Experimental results on benchmark datasets show that feature selection significantly improves

Table 16: Comparative analysis of proposed feature selection method with baseline techniques on healthcare and ant 1.3 datasets

Feature Selection	Metric	Healthcare Dataset						Ant 1.3 Dataset					
		GBR	RFR	BR	ABR	VR	XGBR	GBR	RFR	BR	ABR	VR	XGBR
RATLIFF	MSE	1.86	1.39	1.39	1.45	1.59	1.52	0.024	0.021	0.023	0.034	0.022	0.029
	MAE	1.05	0.91	0.87	1.02	0.97	0.84	0.061	0.057	0.058	0.074	0.058	0.068
	RMSE	1.36	1.18	1.18	1.21	1.26	1.23	0.155	0.144	0.152	0.185	0.148	0.171
	SERA	1.77	1.35	1.38	1.40	1.54	1.43	0.024	0.021	0.023	0.034	0.022	0.029
SVM-RFC	MSE	1.17	1.43	1.58	1.35	1.21	1.18	0.022	0.020	0.027	0.028	0.023	0.032
	MAE	0.84	0.92	0.99	1.02	0.88	0.79	0.058	0.055	0.061	0.067	0.059	0.069
	RMSE	1.08	1.20	1.26	1.16	1.10	1.09	0.151	0.142	0.165	0.167	0.153	0.180
	SERA	1.16	1.39	1.56	1.30	1.19	1.15	0.022	0.020	0.027	0.028	0.023	0.032
SFS	MSE	1.24	1.30	1.43	1.35	1.30	1.24	0.020	0.020	0.020	0.030	0.020	0.040
	MAE	0.89	0.92	0.90	1.05	0.91	0.84	0.050	0.060	0.060	0.070	0.060	0.070
	RMSE	1.11	1.14	1.20	1.16	1.14	1.11	0.130	0.140	0.140	0.170	0.140	0.190
	SERA	1.22	1.26	1.33	1.33	1.26	1.21	0.020	0.020	0.020	0.030	0.020	0.040
LASSO (L1 Regularization)	MSE	1.46	1.51	1.82	1.44	1.39	1.49	1.46	1.51	1.82	1.52	1.52	2.03
	MAE	0.92	0.97	1.12	0.98	0.91	0.93	0.88	0.90	0.92	1.00	0.91	1.01
	RMSE	1.21	1.23	1.35	1.20	1.18	1.22	1.21	1.23	1.35	1.23	1.23	1.43
	SERA	1.45	1.49	1.79	1.40	1.38	1.44	1.45	1.50	1.81	1.50	1.51	2.03
RIDGE (L2 Regularization)	MSE	1.46	1.49	1.67	1.25	1.39	1.49	0.03	0.03	0.03	0.04	0.03	0.04
	MAE	0.92	0.97	0.97	0.94	0.92	0.93	0.07	0.06	0.07	0.07	0.07	0.07
	RMSE	1.21	1.22	1.29	1.12	1.18	1.22	0.19	0.18	0.19	0.19	0.18	0.19
	SERA	1.45	1.47	1.66	1.22	1.37	1.44	0.03	0.03	0.03	0.04	0.03	0.04
MUTUAL INFORMATION	MSE	1.72	1.41	1.61	1.52	1.58	1.38	1.46	1.51	1.82	1.52	1.52	2.03
	MAE	1.07	0.98	1.06	1.04	1.04	0.83	0.88	0.90	0.92	1.00	0.91	1.01
	RMSE	1.31	1.19	1.27	1.23	1.26	1.17	1.21	1.23	1.35	1.23	1.23	1.43
	SERA	1.68	1.38	1.53	1.49	1.54	1.38	1.45	1.50	1.81	1.50	1.51	2.03
TREE-BASED MODELS	MSE	1.46	1.51	1.82	1.52	1.52	2.03	1.46	1.51	1.82	1.52	1.52	2.03
	MAE	0.88	0.90	0.92	1.00	0.91	1.01	0.88	0.90	0.92	1.00	0.91	1.01
	RMSE	1.21	1.23	1.35	1.23	1.23	1.43	1.21	1.23	1.35	1.23	1.23	1.43
	SERA	1.45	1.50	1.81	1.50	1.51	2.03	1.45	1.50	1.81	1.50	1.51	2.03

prediction accuracy compared to models without feature selection. The Wilcoxon Signed-Rank Test further confirms the statistical significance of these improvements.

The proposed FFO was also compared with PSO, GWO, and GA. Results indicate that FFO generally provides higher prediction accuracy and more consistent performance, although with slightly increased computational time. Overall, the proposed framework enhances software defect density prediction in terms of accuracy, robustness, and reliability. Future work will focus on more efficient hybrid optimization methods, large-scale industrial datasets, and explainable AI techniques to improve model interpretability.

6 Validity concerns

This study considers potential threats to validity that may affect the reliability and generalizability of the proposed framework. These threats are categorized as internal, external, construct, and contextual validity.

6.1 Internal validity

Internal validity relates to the correctness of the experimental design and parameter settings. The performance of the FFO-based feature selection method may vary with parameter tuning, regression models, and dataset characteristics. To reduce bias, experiments are conducted under consistent settings using systematic parameter tuning. Multiple regression models and evaluation metrics such as MSE, MAE, RMSE, and R^2 are used for fair comparison.

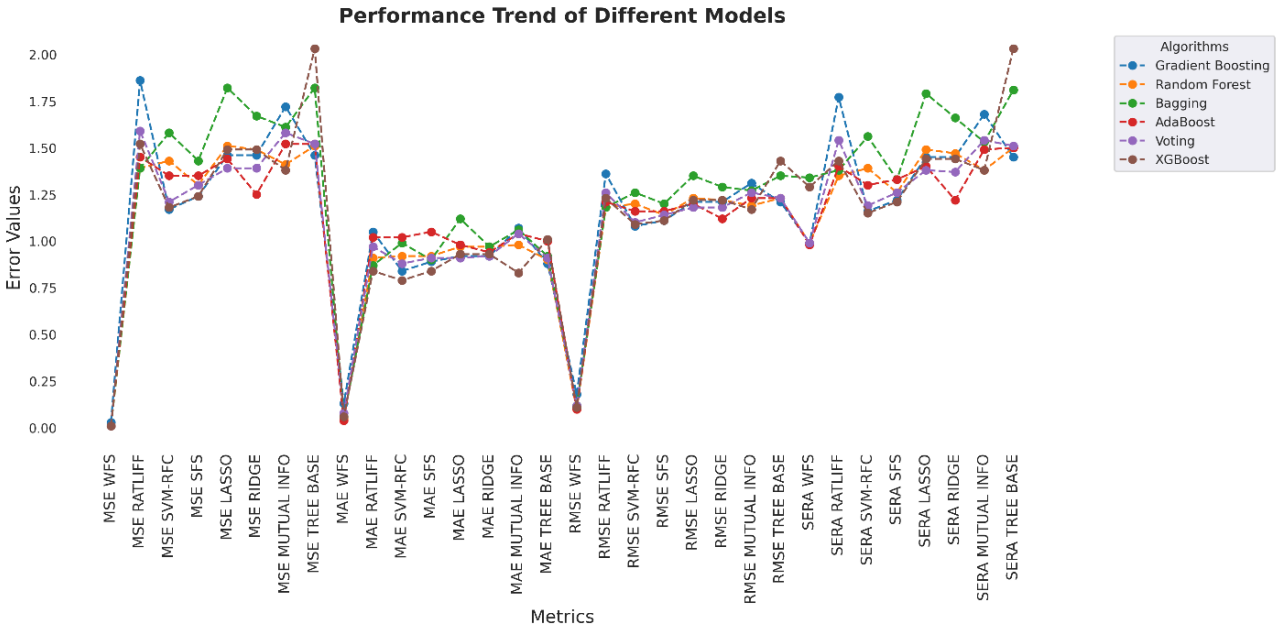
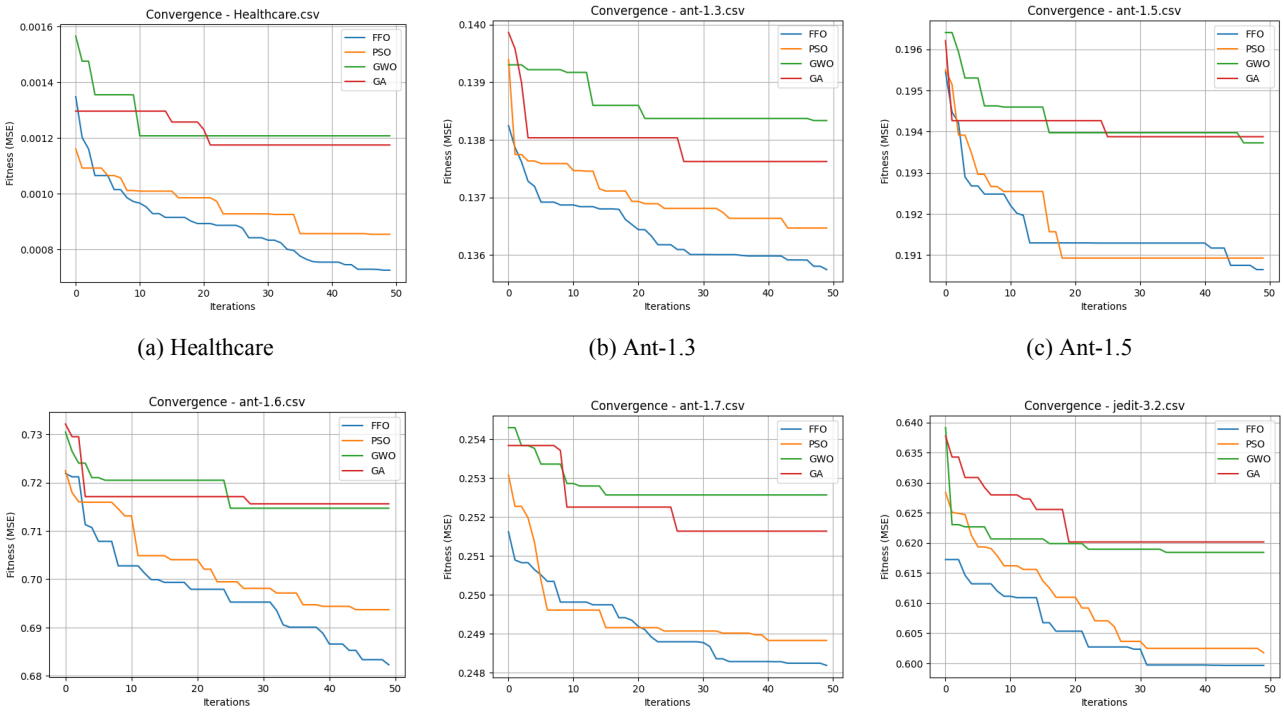


Figure 2: Your caption here

Table 17: Final results of different optimization algorithms across datasets

Dataset	Algorithm	RMSE_mean	RMSE_std	Runtime_mean	Runtime_std
Ant-1.3	FFO	0.439404	0.140196	3.674661	0.961077
	PSO	0.445489	0.128683	2.402033	0.247923
	GWO	0.531487	0.178112	2.295716	0.214090
	GA	0.453674	0.132463	2.157599	0.144698
Ant-1.7	FFO	0.520653	0.039017	2.627413	0.370743
	PSO	0.619243	0.074876	2.520014	0.323196
	GWO	0.627399	0.092002	2.449803	0.273045
	GA	0.604223	0.081154	2.373454	0.231530
Healthcare	FFO	0.094267	0.002558	2.611937	0.335241
	PSO	0.096805	0.002518	2.383401	0.283470
	GWO	0.095077	0.001671	2.306486	0.212808
	GA	0.097065	0.001999	2.251248	0.276943
Ant-1.5	FFO	0.459137	0.007963	2.555871	0.204464
	PSO	0.563468	0.133885	2.465245	0.265747
	GWO	0.574375	0.134621	2.389692	0.329062
	GA	0.568207	0.131319	2.305727	0.224284
Ant-1.6	FFO	0.859752	0.015951	2.790464	0.441122
	PSO	0.884996	0.025528	2.466493	0.239034
	GWO	0.896830	0.025257	2.398887	0.370041
	GA	0.886560	0.013295	2.233055	0.152644
Jedit-3.2	FFO	0.872713	0.065866	2.530309	0.187205
	PSO	0.923146	0.053453	2.394542	0.212634
	GWO	0.891063	0.056567	2.378632	0.327168
	GA	0.844038	0.019496	2.307548	0.224559



(a) Healthcare (b) Ant-1.3 (c) Ant-1.5

(d) Ant-1.6 (e) Ant-1.7 (f) JEdit-3.2

Figure 3: Convergence graphs comparing FFO with PSO, GWO, and GA across different datasets. The proposed FFO algorithm consistently shows faster convergence and lower error values.

Table 18: Parameter settings of optimization algorithms

Parameter	**FFO**	**PSO**	**GWO**	**GA**
Population Size	25	25	25	25
Iterations	50	50	50	50
w	–	0.7	–	–
c_1, c_2	–	1.7, 1.7	–	–
Mutation Rate	–	–	–	0.3

Table 19: Wilcoxon signed-rank test results

Comparison	**p-value**	**Result**
Proposed vs PSO	0.00577	Significant
Proposed vs GWO	0.00158	Significant
Proposed vs GA	0.00761	Significant

6.2 External validity

External validity concerns the generalizability of the results to real-world software projects. Since benchmark datasets are used, the findings may not represent all software environments. To address this issue, experiments are performed on multiple heterogeneous datasets with diverse characteristics, improving the applicability of the proposed approach.

6.3 Construct validity

Construct validity evaluates whether the proposed method accurately measures software defect density prediction performance. Inappropriate baselines or inconsistent preprocessing may affect the results. To mitigate this threat, the proposed approach is compared with established feature selection methods, and uniform preprocessing is applied across all datasets.

6.4 Contextual validity

Contextual validity refers to the impact of evolving software practices, tools, and dataset distributions on model performance. The effectiveness of the proposed method may vary across different environments. To improve robustness, experiments are conducted using multiple regression models and diverse datasets, ensuring adaptability in practical applications.

References

[1] F. Alghanim, M. Azzeh, A. El-Hassan, and H. Qattous, "Software defect density prediction using deep learning," *IEEE Access*, vol. 10, pp. 114 629–114 641, 2022.

[2] C. López-Martín, Y. Villuendas-Rey, M. Azzeh, A. B. Nassif, and S. Banitaan, "Transformed k-nearest

neighborhood output distance minimization for predicting the defect density of software projects," *Journal of Systems and Software*, vol. 167, p. 110592, 2020.

[3] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?" *Software Quality Journal*, vol. 26, pp. 525–552, 2018.

[4] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," in *Proceedings. 26th International Conference on Software Engineering*. IEEE, 2004, pp. 282–291.

[5] S. Rathaur, N. Kamath, and U. Ghanekar, "Software defect density prediction based on multiple linear regression," in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, 2020, pp. 434–439.

[6] V. Kumar, A. Sharma, and R. Kumar, "Applying soft computing approaches to predict defect density in software product releases: an empirical study," *Computing and Informatics*, vol. 32, no. 1, pp. 203–224, 2013.

[7] C. Tantithamthavorn, A. E. Hassan, and K. Matsumoto, "The impact of class rebalancing techniques on the performance and interpretation of defect prediction models," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1200–1219, 2018.

[8] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, pp. 540–578, 2009.

[9] F. Wu, X.-Y. Jing, Y. Sun, J. Sun, L. Huang, F. Cui, and Y. Sun, "Cross-project and within-project semisupervised software defect prediction: A unified approach," *IEEE Transactions on Reliability*, vol. 67, no. 2, pp. 581–597, 2018.

[10] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE transactions on software engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[11] L. L. Minku, "A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation," *Empirical Software Engineering*, vol. 24, no. 5, pp. 3153–3204, 2019.

[12] C. Rahmani and D. Khazanchi, "A study on defect density of open source software," in *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*. IEEE, 2010, pp. 679–683.

[13] N. Mandhan, D. K. Verma, and S. Kumar, "Analysis of approach for predicting software defect density using static metrics," in *International Conference on Computing, Communication & Automation*. IEEE, 2015, pp. 880–886.

[14] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 284–292.

[15] S. K. Khalsa, "A fuzzified approach for the prediction of fault proneness and defect density," in *Proceedings of the World Congress on Engineering*, vol. 1, 2009, pp. 218–223.

[16] O. Kutlubay, B. Turhan, and A. B. Bener, "A two-step model for defect density estimation," in *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007)*. IEEE, 2007, pp. 322–332.

[17] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proceedings of the 2006 international workshop on Mining software repositories*, 2006, pp. 119–125.

[18] C. López-Martín, M. Azzeh, A. Bou-Nassif, and S. Banitaan, "Upsilon-SVR polynomial kernel for predicting the defect density in new software projects," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1377–1382.

[19] Y. Zhang, P. Masci, P. Jones, and H. Thimbleby, "User interface software errors in medical devices: study of US recall data," *Biomedical Instrumentation & Technology*, vol. 53, no. 3, pp. 182–194, 2019.

[20] R. Jetley and B. Chelf, "Diagnosing medical device software defects using static analysis," *Coverity. Published in MD&DI*, 2009.

[21] P. Branco, L. Torgo, and R. P. Ribeiro, "SMOGN: a pre-processing approach for imbalanced regression," in *First international workshop on learning with imbalanced domains: Theory and applications*. PMLR, 2017, pp. 36–50.