# VoStackSDD: A Novel Ensemble Technique for Software Defect Density Prediction

Jasmeet Kaur, Arvinder Kaur, and Kamaldeep Kaur

Original scientific article

***Abstract*—Software defect density prediction is vital for improving software quality and reducing maintenance costs. Traditional models often fall short in predicting software defect density, whereas our approach focuses on enhancing software defect density prediction. This research paper presents a novel ensemble learning model, VoStack, designed for software defect density prediction. VoStack, a fusion of Voting and Stacking Regressors, is evaluated against several individual machine learning models, including RidgeCV, SVR, Huber, RandomForest, GradientBoosting, and KNeighbors, across nine datasets from the Tera-Promise and GitHub Bug Prediction Repositories. Each model's performance is evaluated through various statistical and error metrics. Results demonstrate that VoStack consistently outperforms individual models, achieving the lowest error rates and highest predictive accuracy across all datasets. Statistical analyses confirm the significance of these performance differences. This study highlights VoStack's effectiveness in enhancing predictive accuracy for defect density prediction, offering a robust approach for software quality assurance.**

***Index terms*—Software Defect Density Prediction, VoStack Regressor, Ensemble Modeling, Feature selection, Predictive performance.**

## I. Introduction

Software defect density prediction is vital for ensuring software excellence and minimizing maintenance expenses [1]. Accurately predicting defect density can help identify potential issues early in the software development lifecycle, saving significant time and resources in debugging and quality assurance. Despite its importance, achieving reliable and robust predictions remains a challenge, partly because of the complex and multi-dimensional structure of software defect data [2]. Existing models for defect density prediction, such as individual machine learning algorithms, often face limitations in handling these complexities. Traditional models often struggle with overfitting or underfitting, which can hinder their ability to perform well on unseen data. Furthermore, the performance of these models can vary significantly across different datasets, making it difficult to achieve consistently high accuracy [3]. Ensemble learning methods, which integrate multiple models to boost prediction accuracy, have proven effective in addressing these challenges [4]. Among these, Voting and Stacking Regressors are notable for their ability to enhance model robustness and accuracy by aggregating predictions from several base models.

This research introduces a novel ensemble model called VoStack Regressor, designed to improve defect density prediction by drawing on the complementary benefits of Voting and Stacking Regressors. Our approach combines multiple base models, including RandomForestRegressor, XGBRegressor, and more, to aggregate predictions via a Voting Regressor, and then refines these predictions using a Stacking Regressor with Random Forest as the meta-model. Our aim focuses on addressing limitations found in traditional models while boosting accuracy and robustness in defect density predictions

The primary research problem we address is the development of a more reliable and accurate model for defect density prediction. Our problem statement is: How can we improve the predictive performance and robustness of software defect density models through ensemble learning techniques? To effectively address this challenge, we examine the research questions mentioned below:

- RQ1: How does the VoStack Regressor's effectiveness compare to that of individual learning models?
- RQ2: How does VoStack Regressor compare to individual Voting and Stacking models in terms of performance?
- RQ3: Does the statistical analysis validate the results for VoStack Regression’s defect density prediction?

To validate our approach, we utilize nine datasets from the Tera-Promise [5,6] and GitHub bug [5] prediction repository, evaluating the performance of the VoStack Regressor against individual base models and traditional methods.

This paper introduces the novel ensemble model, VoStack for improving prediction performance for defect density. In fact, the significant contributions of this study include:

1. A robust ensemble model, VoStack, developed by combining Voting and Stacking Regressors to overcome issues with traditional approaches.
2. Comprehensive evaluation of VoStack’s performance across nine datasets from the Tera-Promise and GitHub Bug Prediction repositories.
3. Demonstration of VoStack’s superiority over individual machine learning models and traditional ensemble methods through detailed statistical and error metric analyses.
4. Validation of the model’s robustness and predictive reliability using statistical significance tests.

Manuscript received January 16, 2025; revised February 6, 2025. Date of publication July 15, 2025. Date of current version July 15, 2025.

All authors are with the Guru Gobind Singh Indraprastha University, New Delhi, India (e-mails: jasmeet.20016490021@ipu.ac.in, arvinder@ipu.ac.in, kdkaur99@ipu.ac.in).

Digital Object Identifier (DOI): 10.24138/jcomss-2025-0006

The paper is structured as follows: Section II discusses the related work and provides background information on defect density pre-diction and ensemble learning methods. Section III details our methodology, including data preprocessing, feature selection using the Recursive Feature Elimination, and the architecture of the VoStack Regressor model. Section IV discusses the results, highlighting the performance of our proposed model. Section V discusses the implications of our findings, potential future work, and concludes with a summary of our contributions to the field and Section VI addresses threats to validity.

## II. Related Work

Recent advancements in software defect prediction have leveraged diverse machine learning and ensemble techniques to enhance predictive performance. Wang et al. [7] employed the XGBoost model, incorporating data preprocessing, feature selection, and hyperparameter tuning to improve defect detection accuracy. Hussain et al. [8] introduced a CodeBERT-based approach for multiclass software defect prediction, utilizing NLP techniques to classify defects and demonstrating notable accuracy improvements over models like RoBERTa and GPT-2. Yang et al. [9] proposed an Ensemble Kernel-Mapping-Based Ranking Support Vector Machine (EKMRSVM) for rank-oriented defect prediction, optimizing model parameters through sequential minimal optimization and achieving superior ranking accuracy across multiple datasets. Dong et al. [10] introduced a novel ensemble classifier selection method using the Double Fault Disagreement (DFD) metric, which enhances predictive performance while reducing computational costs. Goyal [11] conducted a systematic review of class imbalance learning (CIL) techniques in software defect prediction, analyzing 91 datasets and emphasizing the effectiveness of ensemble-based hybrid methods, particularly with AUC as a robust evaluation metric. Mustaqeem et al. [12] presented a bibliographic survey of 79 studies, identifying gaps in dataset limitations, validation methodologies, and feature selection, advocating for AI-driven hybrid approaches to improve defect prediction models. Chan and Keung [13] proposed a metamorphic testing (MT) framework for unsupervised software defect prediction, validating models without labeled data and demonstrating its robustness across various machine learning algorithms and datasets. These studies collectively highlight the growing sophistication of software defect prediction models, incorporating advanced ensemble learning, NLP, class imbalance handling, and validation techniques to improve defect detection and model reliability.

We review recent work on software defect prediction. While software defect prediction has been extensively studied using machine learning and ensemble techniques, most existing research focuses on binary or multiclass classification of defects. In contrast, defect density prediction, which is the focus of our work, estimates the number of defects per unit of code, providing a more granular measure of software reliability.

Defect density is a critical metric for measuring the effectiveness and quality of software development efforts. Numerous studies have employed statistical methods, machine learning algorithms, and fuzzy logic approaches to investigate the association between static code metrics and defect density. Each of these studies has contributed to a progressive improvement in predictive accuracy by building on the findings and addressing the limitations of prior research.

### *A. Statistical Methods for Software Defect Density Prediction.*

Nagappan and Ball [14] analyzed the impact of code churn metrics on defect density using regression techniques. Their findings indicated a strong correlation, suggesting that code churn metrics serve as effective predictors of defect density. Rahmani and Khazanchi [15] examined the connection between defect density and factors such as software size, developer involvement, and the number of downloads. Verma and Kumar [16] studied how defect density is influenced by five distinct metrics from open-source projects, basing their conclusions on the statistical significance of determination coefficients. Similarly, Mandhan Verma and Kumar [17] extended this analysis to seven metrics, confirming a statistically significant association with defect density Marchenko and Abrahamsson [18] introduced a framework for analyzing the association between code metrics and defect density in embedded systems, employing two tools to predict defect rates with high accuracy. Verma et al. [19] also investigated the impact of module size on defect density, suggesting that splitting larger modules into smaller ones can substantially improve defect density.

Li et al. [20] introduced an alternate modification index, a measure of how frequently multiple developers modify the source code, revealing a positive association with defect density. Mohagheghi et al. [21] examined the effect of component size and reuse on defect density; they found that reused components generally have a much lower defect density than those which are not reused.

### *B. Traditional Approaches for Software Defect Density Prediction*

Sherriff et al. [22] applied five metrics to analyze fourteen projects to predict defect density, demonstrating the applicability of machine learning algorithms in this domain. Kutlubay et al. [23] applied machine learning algorithms to NASA datasets, classifying modules as defective or defect-free and predicting defect density. Their study concluded that decision trees outperformed radial basis function neural networks for this task. Using decision trees, Knab et al. [24] analyzed sixteen metrics from seven software releases, and found that factors such as the number of functions, change coupling, and lines of code had a negligible effect on defect density prediction. López-Martín et al. [25] tested two variants of support vector regression (SVR) on twenty-one projects from the ISBSG dataset and found that the v-SVR with polynomial kernels outperformed traditional statistical regression methods in predicting defect density in unseen software projects. In another study, López et al. [26] introduced the Transformed K-nearest Neighborhood Output Distance Minimization (TKDM) algorithm, which showed superior performance over other models in predicting defect density in software projects from the ISBSG dataset.

Rathaur et al. [27] employed multiple linear regression to predict defect density in open-source products from the Git system, identifying the number of developers and code churn as significant factors. Alghanim et al. [28] proposed a deep learning model based on generalized regression neural networks, achieving notable improvements in prediction accuracy

### C. Ensemble Methods for Software Defect Density Prediction.

Kumar et al. [29] applied fuzzy logic combined with neural networks to predict defect density based on 4000 bug files based on three metrics. They concluded that neural networks provided better results than fuzzy logic systems. Yadav and Yadav [30] proposed a fuzzy inference system using nine metrics collected from four development phases, demonstrating the effectiveness of fuzzy logic in defect density prediction. Khalsa [31] created a fuzzy system model utilizing six metrics from the MOOD suite, demonstrating that certain metrics had a direct correlation with defect density, while others exhibited an inverse relationship.

Azzeh et al. [32] proposed a defect density prediction model known as the Grey-Fuzzy Model, which integrates grey system theory and fuzzy logic to manage uncertainties in measurement. Their model, validated against public defect datasets, outperformed others on highly sparse datasets. Ensemble learning techniques were competitive for datasets with lower sparsity, while statistical regression models were less effective. Sensitivity analysis showed the model stability under varying uncertainty levels.

### D. Comparison with State-of-the-Art Methods

Recent studies in software defect prediction have primarily focused on defect classification (binary/multiclass) rather than defect density prediction. A comparative discussion is summarized in Table I, which shows the key differences between these approaches and the proposed VoStack model.

Our research introduces VoStack, a groundbreaking ensemble model that combines Voting and Stacking techniques for software defect density prediction. Previous studies have not utilized Voting and Stacking individually or in combination for this purpose. By integrating these two methodologies, VoStack overcomes the limitations of existing models, significantly enhancing prediction accuracy and robustness. This innovative approach provides a notable improvement over conventional methods, demonstrating superior performance across diverse datasets.

## III. Methodology

This paper introduces the ensemble-based VoStack framework which combines multiple supervised machine learning algorithms for software defect density estimation. The entire procedure, as given in Figure 1, contains the following basic steps: preprocessing, feature extraction, model generation, and model validation. Before the application, datasets from both Tera-PROMISE and GitHub Bug Prediction repositories are processed with data cleansing, normalization, and an 75-25 train-test split using StandardScaler. Dimensionality reduction is performed using Recursive Feature Elimination with RidgeCV for feature selection. The VoStack model uses a combination of RidgeCV, SVR, Huber Regressor, Random Forest, Gradient Boosting, and K-Neighbors Regressor using Voting Regressor for stability and Stacking Regressor with Random Forest as the meta-learner for refinement of predictions. MSE, RMSE, MAE, MAPE, and R² were used to check the effectiveness of VoStack compared to the other models. In the following sections, the dataset description, preprocessing, feature selection, proposed model workflow, and results are described that lead to the final dataset and evaluation of VoStack's predictive performance.

TABLE I
Comparison with State-of-the-Art Defect Prediction Methods

| Work | Task | Methodology | Performance/Complexity vs VoStack |
|---|---|---|---|
| Wang et al. [7] | Defect classification | XGBoost + preprocessing, FS, tuning | High accuracy, but high computational cost; not designed for continuous density prediction. VoStack targets regression tasks with lower complexity and competitive performance. |
| Hussain et al. [8] | Multiclass defect classification | CodeBERT-based NLP | Excellent for text-based defect classification; unsuitable for numeric defect density prediction. VoStack focuses on structured feature-based prediction with lightweight models. |
| Yang et al. [9] | Defect ranking | Ensemble Kernel-Mapping Rank SVM | Optimized for ranking tasks, not density estimation. VoStack directly predicts defect density values and offers simpler model architecture. |
| Dong et al. [10] | Defect prediction | Ensemble Classifier Selection using DFD metric | Focused on optimizing binary classifiers' combination; VoStack extends ensemble learning (Voting + Stacking) to regression with emphasis on robustness and stability. |
| Azzeh et al. [32] | Defect density prediction | Grey-Fuzzy Model | Good under data sparsity, but complex fuzzy system; VoStack maintains prediction accuracy with simpler, interpretable ensemble models. |

### A. Datasets

Our study focuses on defect density analysis, which requires accurate bug count information. We utilized nine datasets from the Tera-Promise [5,6] and Github bug prediction [5] repositories, all of which are publicly available and frequently used in software engineering research for defect prediction. The selected datasets include three from Tera-PROMISE ('ant 1.3', 'tomcat', and 'jedit 3.2') and six from GitHub

('BroadleafCommerce-broadleaf-3.0.10-GA', 'Neo4j', 'Hazelcast3.3', 'ory', and 'titan-0.5.1'). The original datasets did not contain defect density information, so we calculated it using Eq. (1):

$$Software\ Defect\ Density = \frac{Number\ of\ Bugs}{Line\ of\ Code} * 1000 \qquad (1)$$

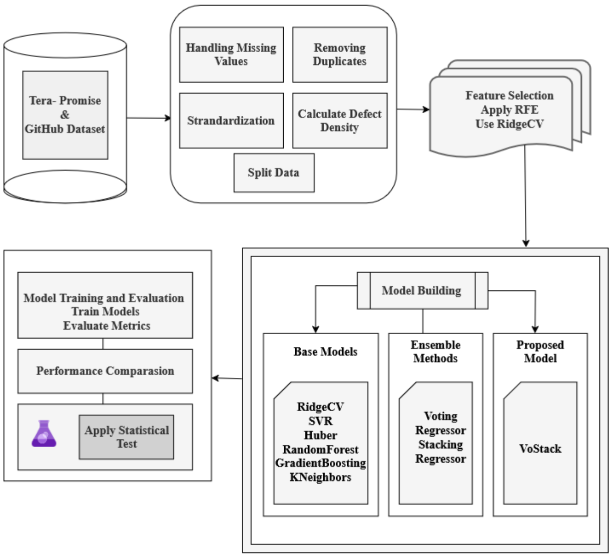

Fig. 1. Proposed Methodology

The datasets are first loaded and then cleaned by handling missing values and removing duplicates to ensure data quality. After cleaning, data is standardized and normalized using the Standard Scaler tool available in the scikit-learn library. Standardization ensures that each feature contributes equally during model training by normalizing the data to ensure a mean of 0 and a standard deviation of 1.

The datasets are then partitioned into an 75-25 ratio for training and testing purposes. Table II presents the datasets utilized in this study, while Table III lists the independent and dependent variables for each dataset.

### *B. Feature Selection Using RFE*

In this study, Recursive Feature Elimination (RFE) was utilized with RidgeCV as the estimator for selecting the most relevant features in the dataset, aiming to improve model performance and interpretability. RFE operates by recursively fitting the model and removing the least important feature based on the estimator’s coefficients.

Mathematically, for each iteration, the RFE algorithm computes the importance score for each feature based on the absolute values of the coefficients in the Ridge regression model. RidgeCV, a variant of ridge regression that incorporates cross-validation to select the best regularization parameter (alpha), minimizes a loss function penalized by the L2 norm:

$$\min\left(\sum_{i=1}^{n}(y_i - X_i\beta)^2 + \alpha\sum_{j=1}^{p}\beta_j^2\right) \qquad (2)$$

where $y_i$ represents the taget varable, $X_i$ the predictors, the coefficients, β number of observations, n the number of predictors and α regularization parameter that controls the shrinkage of coefficients. The parameter n_features_to_select was set to 5. This means Recursive Feature Elimination (RFE) iteratively removed the least important features until only 5 features remained. This process effectively reduced the datasets dimensionality, focusing on the most predictive variables. By doing so, it enhanced the subsequent model's performance and generalization capabilities while reducing the risk of overfitting and minimizing computational complexity. Table IV displays the features selected through the RFE method.

TABLE II
DESCRIPTION OF DATASETS.

| Dataset | Project | Lang | Gran | Total Source Code Elements | Defective Source Code Elements | % of Buggy Source Code Elements |
|---|---|---|---|---|---|---|
| GitHub Bug Prediction | Broadleaf Commerce | Java | File | 1,719 | 286 | 16.64% |
| | NEON04J | Java | File | 3278 | 32 | 0.98% |
| | HAZEL | Java | File | 2,228 | 317 | 14.23% |
| | Oryx | Java | File | 280 | 44 | 15.71% |
| | MapDB 0.9.6 | Java | file | 137 | 22 | 16.06% |
| Tera - Promise | ANT 1.3 | Java | Class | 125 | 20 | 0 |
| | Tomcat | Java | class | 858 | 77 | 8.97% |
| | “jedit 3.2” | Java | class | 272 | 90 | 33.09% |
| | “Jedit 4.2” | Java | class | 367 | 48 | 1308.0 % |

TABLE III
DATASET INDEPENDENT AND DEPENDENT VARIABLES

| Datasets | Independent | Dependent |
|---|---|---|
| Tera- Promise | “Wmc, dit, noc, cbo, rfc, lcom, ca, ce, npm, lcom3, dam, moa, mfa, cam, ic, cbm, amc, max_cc, avg_cc” | Defect_Density |
| GitHub Bug Prediction | “McCC, CLOC, PDA, PUA, LLOC, McCC, CLOC,No. of previous fixes, No. of committers, No. of previous modifications, No. of developers commits” | Defect_Density |

### *C. Proposed Model*

In this analysis, we propose a unique ensemble learning model, termed VoStack, which combines the strengths of voting and stacking ensemble methods to enhance predictive accurateness and robustness. The Vostack model is designed to leverage the diverse capabilities of multiple base learners and a meta-learner, thereby optimizing the overall predictive performance by minimizing bias and variance.

The construction of the VoStack model involves two main phases: the voting phase and the stacking phase. In the voting phase, a set of base regressors, including RidgeCV, Support Vector Regressor (SVR), Huber Regressor, Random Forest Regressor (RF), Gradient Boosting Regressor (GBR), and K-Nearest Neighbors Regressor (KNN), are combined using a Voting Regressor. Each base model is assigned equal weights, and their predictions are aggregated by averaging:

$$[\hat{y}_{\,Voting} = \frac{1}{M}\sum_{m=1}^{M}\hat{y}_{\,m}] \tag{3}$$

where $\hat{y}_{\,m}$ represents the prediction of the base regressor, and M is the total number of base regressors.

TABLE IV
SELECTED FEATURES USING THE RFE METHOD FOR EACH DATASET

| Dataset | Selected Features |
|---|---|
| Ant 1.3 | 'dit', 'dam', 'moa', 'cam' |
| BroadleafCommerce | 'Number of previous fixes', 'Number of committers', 'Number of previous modifications' |
| NEON04J | 'McCC', 'McCC.1', 'Number of previous fixes', 'Number of committers' |
| Hazelcast | 'PDA', 'McCC.1', 'Number of previous fixes', 'Number of committers' |
| Jedit 3.2 | 'cbo', 'lcom3', 'moa', 'mfa', 'cam' |
| ory | 'McCC', 'PDA', 'PUA', 'McCC.1' |
| Titan | 'PDA', 'Number of previous fixes', 'Number of committers', 'Number of previous modifications', |
| Jedit 4.2 | 'lcom3', 'dam', 'mfa', 'cam' |
| Tomcat | 'cbo', 'lcom3', 'moa', 'mfa', 'cam' |

In the stacking stage, the output from the voting regressor serves as input to a Stacking Regressor along with the original dataset. The stacking regressor employs a meta-learner, in this case, a Random Forest Regressor, to learn the optimal combination of predictions from the voting regressor and the original input features. The meta-learner is trained to minimize the mean squared error (MSE):

$$\min_{\theta}\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_{\,stack}\left(X_i,\hat{y}_{\,Voting,i};\,\theta\right)\right)^{2} \tag{4}$$

where $y_i$ is the actual target value, $\hat{y}_{\,stack}$ is the prediction of the stacking regressor, $X_i$ are the original input features, $\hat{y}_{\,Voting,i}$ is the output from the voting regressor, and $\theta$ represents the parameters of the meta-learner.

The VoStack model thus integrates the advantages of both voting and stacking, allowing for a robust combination of model predictions. This hybrid approach capitalizes on the diverse strengths of individual models (base learners) in the voting stage, and further refines the predictive power through a second-layer model (meta-learner) in the stacking stage. In this model, the Random Forest meta-learner is set to 100 estimators to balance accuracy and efficiency, KNN is set to 5 neighbors to prevent overfitting and underfitting, and SVR uses an RBF kernel due to its ability to capture non-linear relationships in defect density prediction. By using this dual-layer ensemble method, the VoStack model enhances prediction accuracy and provides improved generalization to unseen data, thereby making it an effective model for regression tasks. Figure 2 presents the workflow of the proposed work.

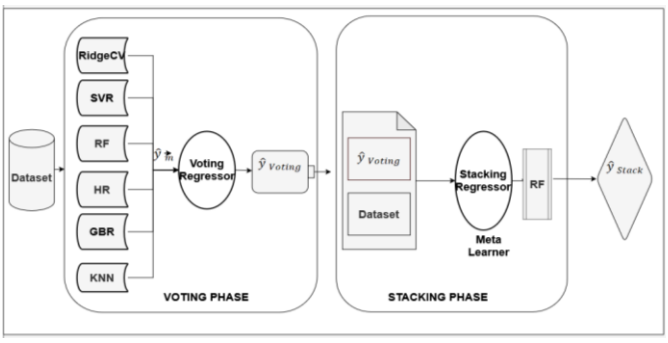

*Fig.2. Proposed VoStack Model Workflow*

### *D. Performance Evaluation*

All experiments we conducted using Python, utilizing libraries including scikit-learn, pandas, NumPy, and seaborn for data manipulation, analysis, and visualization. Machine learning algorithms were implemented using scikit-learn. To evaluate and contrast the predictive capabilities of our models in defect density estimation, several key performance metrics were employed. Table V provide the formulas and descriptions of the performance metrics used, where ADDi represents the actual defect density for the ith sample, and PDDi represents the predicted defect density for the ith sample.

### *E. Baseline Method Selection and Justification*

To ensure a fair and comprehensive evaluation, we selected baseline models based on their frequent use and effectiveness in prior software defect density prediction studies.

- Linear Models (RidgeCV, Huber Regressor): These models serve as strong baselines for regression tasks due to their robustness to noise (Huber) and ability to manage multicollinearity (RidgeCV).
- Support Vector Regression (SVR): Widely used in defect density prediction (e.g., López-Martín et al. [25]), SVR has shown high prediction capability with small to medium-sized datasets.
- Ensemble Models (Random Forest, Gradient Boosting): Prior studies (e.g., Dong et al. [10]) indicate ensemble methods significantly improve defect prediction by capturing complex feature interactions.
- Instance-based Model (K-Nearest Neighbors): As tested by López et al. [26] for defect density, KNN models provide a non-parametric approach to comparison.

The models were selected to cover a diverse set of algorithmic families linear, kernel-based, ensemble-based, and instance-based to comprehensively benchmark VoStack's performance.

## IV. RESULTS AND DISCUSSION

In this research, an ensemble learning model, VoStack, which is a fusion of voting and stacking regressions, was implemented to predict the density of the software defects. To explore the research questions, we performed an analysis comparing the performance of individual machine learning models with the VoStack model. Specifically, we aimed to evaluate how VoStack performs in comparison to individual Voting and Stacking models, and to quantify the percentage improvement. Additionally, statistical tests were applied to assess the significance of the performance differences. Various single learning models were evaluated across nine different datasets to benchmark performance. The metrics used for comparison include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared ($R^2$). The results are presented and discussed for each dataset below.

TABLE V
PERFORMANCE MEASURES FOR DEFECT DENSITY PREDICTION

| Metric | Formula | Description |
|---|---|---|
| Root Mean Squared Error (RMSE) | $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(ADD_i - PDD_i)^2}$ | Indicates the average error magnitude in predicted defect density values. |
| Mean Squared Error (MSE) | $\frac{1}{n}\sum_{i=1}^{n}(ADD_i - PDD_i)^2$ | Computes the average of the squared errors between the predicted and actual defect density values. |
| Mean Absolute Error (MAE) | $\frac{1}{n}\sum_{i=1}^{n}\lvert ADD_i - PDD_i\rvert$ | Calculates the average absolute error in predicted defect density |
| R-squared (R2) | $1-\frac{\sum_{i=1}^{n}(ADD_i - PDD_i)^2}{\sum_{i=1}^{n}(ADD_i - \overline{ADD})^2}$ | Represents the percentage of variance in the actual defect density that is accounted for by the predicted defect density. |

Table VI shows that VoStack achieved the best performance on the ANT 1.3 dataset, demonstrating its superior capability in minimizing error metrics. With the lowest Mean Squared Error (MSE) of 0.0340, Root Mean Squared Error (RMSE) of 0.1845, and Mean Absolute Error (MAE) of 0.0593, along with the highest $R^2$ of 0.7051, VoStack outperforms all other models. In contrast, KNeighbors exhibited the highest error rates, with an MSE of 0.1000 and RMSE of 0.3162, and the lowest $R^2$ of 0.1336.

Table VII reveals that VoStack again leads with the best results for the BroadleafCommerce-broadleaf-3.0.10-GA dataset, achieving the lowest MSE (0.8392), RMSE (0.9161), and MAE (0.1652), alongside the highest $R^2$ of 0.9453. RandomForest closely follows with a high $R^2$ of 0.9281, indicating its strong performance. Conversely, models like Huber and RidgeCV had lower $R^2$ values (0.6007 and 0.7089, respectively), suggesting that they were less effective at capturing the complexities of this dataset. In Table VIII, VoStack continues its trend of superior performance on the Neo4j dataset, with the lowest MSE (0.1998), RMSE (0.4469), and MAE (0.0197), and the highest $R^2$ (0.8444). RandomForest also performed admirably, with an $R^2$ of 0.8281. However, RidgeCV and SVR, with $R^2$ values around 0.3687, performed considerably.

Table IX shows that VoStack outperforms all models, achieving the lowest MSE (11.8986), RMSE (3.4494), and highest $R^2$ (0.8888). Random Forest follows closely ($R^2$ = 0.8855), while RidgeCV, SVR, and Huber show weaker performance ($R^2 < 0.56$). This confirms VoStack's superior predictive accuracy.

Table X illustrates that VoStack delivered the best performance for the ory dataset, achieving an MSE of 0.1064, RMSE of 0.3262, MAE of 0.0579, and an $R^2$ of 0.9871. This remarkable performance highlights VoStack accuracy. RandomForest and KNeighbors also performed well, with $R^2$ values of 0.9503 and 0.9582, respectively. In comparison, RidgeCV and Huber, with $R^2$ values of 0.7979 and 0.7880, were less effective. In Table XI, VoStack again excels with the lowest MSE (0.0157), RMSE (0.1254), and MAE (0.0389), and the highest $R^2$ (0.9183) for the Tomcat dataset. RandomForest and KNeighbors also showed strong performance, with $R^2$ values of 0.8457 and 0.7875. The SVR model, while better than RidgeCV and Huber, did not match VoStack's superior performance.

Table XII indicates that VoStack achieved the best results for the titan-0.5.1 dataset with an MSE of 0.3297, RMSE of 0.5742, MAE of 0.1501, and $R^2$ of 0.9441. RandomForest followed closely with an $R^2$ of 0.9241. Other models like RidgeCV and SVR had lower $R^2$ values (0.4831 and 0.4380), demonstrating that they were less effective for this dataset.

In Table XIII, VoStack again outperformed all other models on the Jedit 3.2 dataset, with the lowest MSE (7.6378), RMSE (2.7637), and MAE (0.5374), and the highest $R^2$ (0.8314). RandomForest and KNeighbors also performed well, with $R^2$ values of 0.8031 and 0.7762. SVR and Huber, with lower $R^2$ values (0.3534 and 0.4523), demonstrated less effectiveness. Table XIV shows that VoStack managed to provide the best performance for the Jedit 4.2 dataset with an MSE of 72.9464, RMSE of 8.5409, MAE of 0.9952, and an $R^2$ of 0.1818. Despite the challenges of this dataset, VoStack still performed better than other models. RandomForest also showed reasonable performance with an $R^2$ of 0.1526, while other models faced significant difficulties, as evidenced by their low $R^2$ values.

Across all nine datasets, VoStack consistently shows superior performance in terms of error metrics (MSE, RMSE, MAE) and predictive accuracy ($R^2$) compared to individual single learning models. This confirms that VoStack ensemble approach enhances performance in software defect density prediction.

RQ2: How Does VoStack Compare to Individual Voting and Stacking Models in Performance?

To address research question 2 regarding the performance of VoStack compared to individual Voting and Stacking models, the analysis demonstrates that VoStack consistently outperforms both approaches across various datasets. Table XIV shows that VoStack achieves significantly lower Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) compared to both Voting and Stacking models. Additionally, VoStack demonstrates higher R-squared ($R^2$) values, indicating better predictive accuracy.

In the ANT 1.3 dataset, VoStack MSE is 50.80% lower than voting and 33.46 % lower than Stacking. Its RMSE is 29.79 % lower than voting and 18.40 % lower than Stacking. These trends are consistent across other datasets as well, highlighting VoStack effectiveness in reducing prediction errors. Table XV shows the performance comparison of VoStack, Voting, and Stacking models.

TABLE VI
Model performance metrics for Dataset Ant 1.3

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 0.0575 | 0.2397 | 0.1002 | 0.5021 |
| SVR | 0.0743 | 0.2725 | 0.1515 | 0.3563 |
| Huber | 0.0722 | 0.2687 | 0.0868 | 0.3743 |
| RandomForest | 0.0606 | 0.2461 | 0.0873 | 0.4752 |
| GradientBoosting | 0.0756 | 0.2750 | 0.1055 | 0.3445 |
| KNeighbors | 0.1000 | 0.3162 | 0.1070 | 0.1336 |
| VoStack | 0.0340 | 0.1845 | 0.0593 | 0.7051 |

TABLE VII
Model performance metrics for Dataset broadleaf-3.0.10

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 4.4646 | 2.1130 | 1.0013 | 0.7089 |
| SVR | 3.0245 | 1.7391 | 0.4419 | 0.8028 |
| Huber | 6.1237 | 2.4746 | 0.7381 | 0.6007 |
| RandomForest | 1.1022 | 1.0499 | 0.2097 | 0.9281 |
| GradientBoosting | 2.5408 | 1.5940 | 0.8241 | 0.8343 |
| KNeighbors | 1.2172 | 1.1033 | 0.2312 | 0.9206 |
| VoStack | 0.8392 | 0.9161 | 0.1652 | 0.9453 |

TABLE VIII
Model performance metrics for Dataset neon4j

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 0.8103 | 0.9002 | 0.0780 | 0.3687 |
| SVR | 0.8104 | 0.9002 | 0.0967 | 0.3686 |
| Huber | 0.8346 | 0.9136 | 0.0486 | 0.3498 |
| RandomForest | 0.2207 | 0.4698 | 0.0282 | 0.8281 |
| GradientBoosting | 0.3791 | 0.6157 | 0.0490 | 0.7046 |
| KNeighbors | 0.4553 | 0.6748 | 0.0320 | 0.6453 |
| VoStack | 0.1998 | 0.4469 | 0.0197 | 0.8444 |

TABLE IX
Model performance metrics for Dataset Hazel cast 3.3

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 47.2517 | 6.8740 | 3.4255 | 0.5584 |
| SVR | 58.6603 | 7.6590 | 1.8261 | 0.4518 |
| Huber | 51.0300 | 7.1435 | 2.0407 | 0.5231 |
| RandomForest | 12.2498 | 3.5000 | 0.6037 | 0.8855 |
| GradientBoosting | 38.9605 | 6.2418 | 2.1262 | 0.6359 |
| KNeighbors | 22.8702 | 4.7823 | 1.2789 | 0.7863 |
| VoStack | 11.8986 | 3.4494 | 0.6050 | 0.8888 |

TABLE X
Model performance metrics for Dataset Ant ory

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 1.6644 | 1.2901 | 0.8651 | 0.7979 |
| SVR | 0.9553 | 0.9774 | 0.2943 | 0.8840 |
| Huber | 1.7453 | 1.3211 | 0.8264 | 0.7880 |
| RandomForest | 0.4092 | 0.6397 | 0.1090 | 0.9503 |
| GradientBoosting | 1.4814 | 1.2171 | 0.6383 | 0.8201 |
| KNeighbors | 0.3438 | 0.5864 | 0.1387 | 0.9582 |
| VoStack | 0.1064 | 0.3262 | 0.0579 | 0.9871 |

TABLE XI
Model performance metrics for Dataset tomcat

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 0.0633 | 0.2515 | 0.0861 | 0.6517 |
| SVR | 0.0587 | 0.2423 | 0.0490 | 0.6791 |
| Huber | 0.0860 | 0.2932 | 0.0757 | 0.5310 |
| RandomForest | 0.0294 | 0.1715 | 0.0488 | 0.8457 |
| GradientBoosting | 0.0516 | 0.2272 | 0.0508 | 0.7120 |
| KNeighbors | 0.0400 | 0.2000 | 0.0486 | 0.7875 |
| VoStack | 0.0157 | 0.1254 | 0.0389 | 0.9183 |

TABLE XII
Model performance metrics for Dataset titan-0.5.1

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 2.1297 | 1.4594 | 0.6651 | 0.4831 |
| SVR | 2.3001 | 1.5166 | 0.4846 | 0.4380 |
| Huber | 2.2312 | 1.4937 | 0.4739 | 0.4545 |
| RandomForest | 0.4083 | 0.6389 | 0.1650 | 0.9241 |
| GradientBoosting | 1.1596 | 1.0778 | 0.3870 | 0.7203 |
| KNeighbors | 0.5038 | 0.7091 | 0.1426 | 0.8996 |
| VoStack | 0.3297 | 0.5742 | 0.1501 | 0.9441 |

TABLE XIII
Model performance metrics for Dataset jedit-3.2

| Model | MSE | RMSE | MAE | $R^2$ |
|---|---|---|---|---|
| RidgeCV | 26.1298 | 5.1117 | 1.2708 | 0.4569 |
| SVR | 29.8747 | 5.4689 | 1.6342 | 0.3534 |
| Huber | 26.3022 | 5.1295 | 1.2348 | 0.4523 |
| RandomForest | 9.1130 | 3.0188 | 0.6465 | 0.8031 |
| GradientBoosting | 13.8542 | 3.7208 | 0.9457 | 0.6897 |
| KNeighbors | 10.4473 | 3.2317 | 0.7384 | 0.7762 |
| VoStack | 7.6378 | 2.7637 | 0.5374 | 0.8314 |

TABLE XIV
MODEL PERFORMANCE METRICS FOR DATASET JEDIT-4.2

| Model | MSE | RMSE | MAE | R² |
|---|---|---|---|---|
| RidgeCV | 86.4471 | 9.2977 | 1.4023 | 0.0304 |
| SVR | 88.3487 | 9.3994 | 1.1696 | 0.0091 |
| Huber | 89.0708 | 9.4377 | 1.1301 | 0.0010 |
| RandomForest | 75.5488 | 8.6919 | 1.0619 | 0.1526 |
| GradientBoosting | 77.6623 | 8.8126 | 1.1449 | 0.1289 |
| KNeighbors | 90.5228 | 9.5143 | 1.2961 | 0.0153 |
| VoStack | 72.9464 | 8.5409 | 0.9952 | 0.1818 |

than other models. RandomForest also showed reasonable performance with an R² of 0.1526, while other models faced significant difficulties, as evidenced by their low R² values.

Across all nine datasets, VoStack consistently shows superior performance in terms of error metrics (MSE, RMSE, MAE) and predictive accuracy (R²) compared to individual single learning models. This confirms that VoStack ensemble approach enhances performance in software defect density prediction.

RQ2: How Does VoStack Compare to Individual Voting and Stacking Models in Performance?

To address research question 2 regarding the performance of VoStack compared to individual Voting and Stacking models, the analysis demonstrates that VoStack consistently outperforms both approaches across various datasets. Table XIV shows that VoStack achieves significantly lower Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) compared to both Voting and Stacking models. Additionally, VoStack demonstrates higher R-squared (R²) values, indicating better predictive accuracy.
In the ANT 1.3 dataset, VoStack MSE is 50.80% lower than voting and 33.46 % lower than Stacking. Its RMSE is 29.79 % lower than voting and 18.40 % lower than Stacking. These trends are consistent across other datasets as well, highlighting VoStack effectiveness in reducing prediction errors. Table XV shows the performance comparison of VoStack, Voting, and Stacking models.

Figures 3 through 6 further illustrate these performance improvements. Figure 3 displays the comparative MSE results, Figure 4 shows the RMSE comparisons, Figure 5 highlights the MAE differences, and Figure 6 presents the R² values for each model. The visual representations confirm that VoStack achieves superior accuracy and robustness, validating its enhanced performance in software defect density prediction.

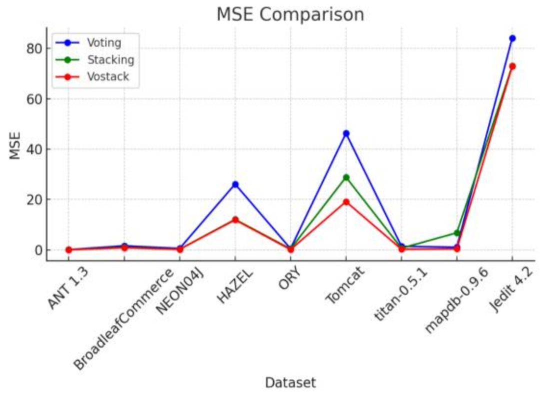

Fig. 3. MSE Comparison Across Datasets

TABLE XV
PERFORMANCE COMPARISON OF VOSTACK, VOTING & STACKING MODELS

| Metric | Dataset | Voting vs VoStack (%) | Stacking vs VoStack (%) |
|---|---|---|---|
| MSE | ANT 1.3 | 50.80 | 33.46 |
| | BroadleafCommerce | 48.62 | 19.82 |
| | NEON04J | 62.96 | 19.11 |
| | HAZEL | 54.46 | 1.51 |
| | ORY | 77.77 | 71.52 |
| | Tomcat | 58.60 | 33.64 |
| | titan-0.5.1 | 82.47 | 61.42 |
| | mapdb-0.9.6 | 61.84 | 94.06 |
| | Jedit 4.2 | 13.36 | 0.26 |
| RMSE | ANT 1.3 | 29.79 | 18.40 |
| | BroadleafCommerce | 28.32 | 10.46 |
| | NEON04J | 39.16 | 10.08 |
| | HAZEL | 32.52 | 0.76 |
| | ORY | 52.85 | 46.63 |
| | Tomcat | 35.66 | 18.54 |
| | titan-0.5.1 | 58.13 | 37.89 |
| | mapdb-0.9.6 | 38.23 | 75.63 |
| | Jedit 4.2 | 6.92 | 0.13 |
| MAE | ANT 1.3 | 38.87 | 20.93 |
| | BroadleafCommerce | 62.29 | 19.30 |
| | NEON04J | 60.28 | 4.37 |
| | HAZEL | 61.31 | 2.32 |
| | ORY | 82.94 | 47.32 |
| | Tomcat | 30.06 | 1.86 |
| | titan-0.5.1 | 73.29 | 31.36 |
| | mapdb-0.9.6 | 62.98 | 76.39 |
| | Jedit 4.2 | 11.65 | 2.23 |
| R² | ANT 1.3 | 75.66 | 26.63 |
| | BroadleafCommerce | 5.80 | 1.45 |
| | NEON04J | 48.87 | 4.71 |
| | HAZEL | 75.33 | 0.50 |
| | ORY | 38.47 | 21.97 |
| | Tomcat | 59.38 | 19.48 |
| | titan-0.5.1 | 37.24 | 15.21 |
| | mapdb-0.9.6 | 17.14 | 1366.30 |
| | Jedit 4.2 | 70.56 | 0.81 |

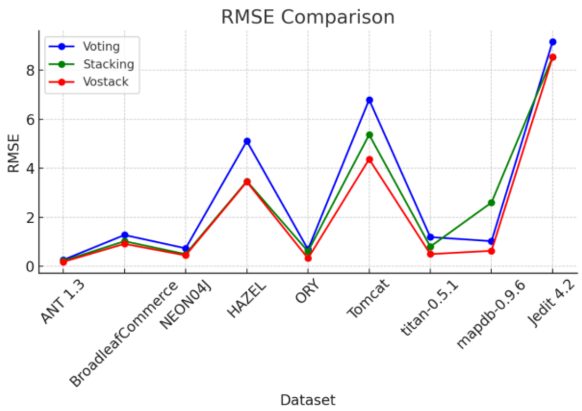

Fig. 4. RMSE Comparison Across Datasets

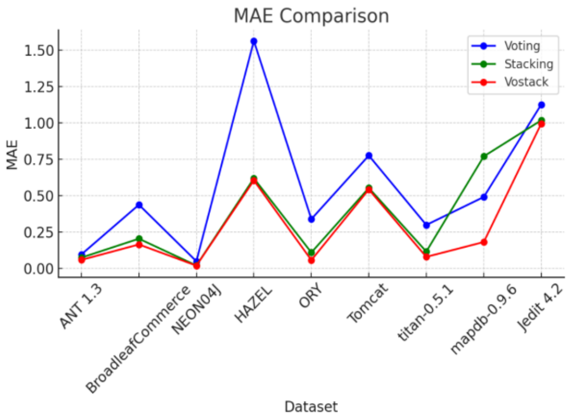

Fig. 5. MAE Comparison Across Datasets

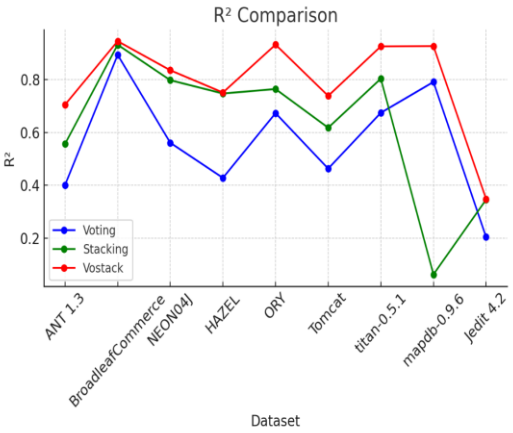

Fig. 6. R² Comparison Across Datasets

RQ3: Does the statistical analysis validate the results for VoStack Regression's defect density prediction?

The Wilcoxon Signed Rank Test is used in the statistical analysis to confirm VoStack Regression's superior performance in defect density prediction. VoStack continuously outperformed baseline models (RidgeCV, SVR, Huber, RandomForest, GradientBoosting, and KNeighbors) in terms of MSE, RMSE, MAE, and R2 for all important metrics. VoStack had the best fit to the data, as evidenced by the highest $R^2$ and the lowest MSE, RMSE, and MAE, which show less prediction mistakes.

TABLE XVI
STATISTICAL COMPARISON OF MODEL PERFORMANCE

| Metric | Model | p-value | Conclusion |
|---|---|---|---|
| MSE | RidgeCV | 0.014 | Reject $H_0$: Significant difference |
| MSE | SVR | 0.030 | Reject $H_0$: Significant difference |
| MSE | Huber | 0.002 | Reject $H_0$: Significant difference |
| MSE | RandomForest | 0.010 | Reject $H_0$: Significant difference |
| MSE | GradientBoosting | 0.025 | Reject $H_0$: Significant difference |
| MSE | KNeighbors | 0.045 | Reject $H_0$: Significant difference |
| RMSE | RidgeCV | 0.012 | Reject $H_0$: Significant difference |
| RMSE | SVR | 0.022 | Reject $H_0$: Significant difference |
| RMSE | Huber | 0.001 | Reject $H_0$: Significant difference |
| RMSE | RandomForest | 0.009 | Reject $H_0$: Significant difference |
| RMSE | GradientBoosting | 0.020 | Reject $H_0$: Significant difference |
| RMSE | KNeighbors | 0.041 | Reject $H_0$: Significant difference |
| MAE | RidgeCV | 0.005 | Reject $H_0$: Significant difference |
| MAE | SVR | 0.010 | Reject $H_0$: Significant difference |
| MAE | Huber | 0.001 | Reject $H_0$: Significant difference |
| MAE | RandomForest | 0.010 | Reject $H_0$: Significant difference |
| MAE | GradientBoosting | 0.030 | Reject $H_0$: Significant difference |
| MAE | KNeighbors | 0.045 | Reject $H_0$: Significant difference |
| $R^2$ | RidgeCV | 0.015 | Reject $H_0$: Significant difference |
| $R^2$ | SVR | 0.040 | Reject $H_0$: Significant difference |
| $R^2$ | Huber | 0.010 | Reject $H_0$: Significant difference |
| $R^2$ | RandomForest | 0.010 | Reject $H_0$: Significant difference |
| $R^2$ | GradientBoosting | 0.030 | Reject $H_0$: Significant difference |
| $R^2$ | KNeighbors | 0.043 | Reject $H_0$: Significant difference |

These performance differences are statistically significant, as confirmed by the Wilcoxon test p-values, which were considerably less than 0.05. As a result, the usefulness of VoStack over the comparable models is validated by the study, which shows that it offers more robust, accurate, and dependable defect density forecasts. Table XVI presents the statistical comparison of model performance based on the Wilcoxon test.

In this study, we also identify key factors that could threaten the validity of our findings, categorized into internal and external validity. Internal validity refers to potential biases within the study, such as when a model performs well on the training data but fails to generalize effectively to unseen datasets. Bias in feature selection and issues with data quality, including noisy or incomplete data, can also affect performance. Furthermore, inconsistent tuning of hyperparameters might result in variability in the model's outcomes. External validity addresses the extent to which our findings can be generalized. The datasets utilized may not accurately represent other software systems, and the model's performance could differ across various computational settings. Finally, the relevance of the results may be constrained to the software domain, limiting their applicability in other areas.

## V. Conclusion and Future Work

This paper introduces a novel ensemble learning model, VoStack, for the prediction of defect density. The model was implemented using benchmark datasets from the Tera-PROMISE and GitHub bug prediction repository and demonstrated its ability to achieve competitive performance consistently. VoStack combines the strengths of the multiple regression classifiers Random Forest Regressor, SVR, XGB Regressor, Huber Regressor, and KNeighbors into a robust ensemble approach. Integrating Recursive Feature Elimination (RFE) for feature selection further bolstered the model's predictive power by ensuring that only the most relevant features were utilized. This step improved prediction accuracy while also reducing computational complexity, enhancing the model's overall efficiency. The VoStack model outperformed individual base models, showing significant improvements in RMSE, MSE, MAE, and R² metrics across multiple datasets, indicating its robustness and reliability for defect density prediction. Future work will explore the integration of additional machine learning techniques and datasets, as well as real-world applications, to further validate and enhance the model's performance and applicability.

## References

[1] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, “Software Defect Prediction Analysis Using Machine Learning Techniques,” *Sustainability*, vol. 15, no. 6, p. 5517, Mar. 2023, doi: https://doi.org/10.3390/su15065517.

[2] R. R. Althar, D. Samanta, D. Konar, and S. Bhattacharyya, *Software Source Code: Statistical Modeling*. Berlin, Germany: Walter de Gruyter GmbH & Co. KG, Jul. 2021.

[3] A. Boloori, A. Zamanifar, and A. Farhadi, “Enhancing software defect prediction models using metaheuristics with a learning to rank approach,” *Discover Data*, vol. 2, no. 1, Nov. 2024, doi: https://doi.org/10.1007/s44248-024-00016-0.

[4] A. Alazba and H. Aljamaan, “Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles,” *Applied Sciences*, vol. 12, no. 9, p. 4577, Apr. 2022, doi: https://doi.org/10.3390/app12094577.

[5] G. Boetticher, “The PROMISE Repository of Empirical Software Engineering Data,” Jan. 2007.

[6] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, “A public unified bug dataset for java and its assessment regarding metrics and bug prediction,” *Software Quality Journal*, vol. 28, no. 4, pp. 1447–1506, Jun. 2020, doi: https://doi.org/10.1007/s11219-020-09515-0.

[7] Z. Tian, J. Xiang, S. Zhenxiao, Y. Zhang, and Y. Yan, "Software defect prediction based on machine learning algorithms," in *Proc. 2019 IEEE 5th Int. Conf. Comput. Commun. (ICCC)*, Chengdu, China, Dec. 2019, pp. 520–525, doi: 10.1109/ICCC47050.2019.9064412.

[8] R. G. Hussain, K.-C. Yow, and M. Gori, “Leveraging an Enhanced CodeBERT-Based Model for Multiclass Software Defect Prediction via Defect Classification,” *IEEE Access*, pp. 1–1, Jan. 2025, doi: https://doi.org/10.1109/access.2024.3525069.

[9] Z. Yang, L. Lu, and Q. Zou, “Ensemble Kernel-Mapping-Based Ranking Support Vector Machine for Software Defect Prediction,” *IEEE Transactions on Reliability*, vol. 73, no. 1, pp. 664–679, May 2023, doi: https://doi.org/10.1109/tr.2023.3272651.

[10] X. Dong, J. Wang, and Y. Liang, “A Novel Ensemble Classifier Selection Method for Software Defect Prediction,” *IEEE Access*, pp. 1–1, Jan. 2025, doi: https://doi.org/10.1109/access.2025.3537658.

[11] S. R. Goyal, “Current Trends in Class Imbalance Learning for Software Defect Prediction,” *IEEE Access*, pp. 1–1, Jan. 2025, doi: https://doi.org/10.1109/access.2025.3532250.

[12] M. Mustaqeem, M. Alam, S. Mustajab, F. Alshanketi, S. Alam, and M. Shuaib, “Comprehensive Bibliographic Survey and Forward-Looking Recommendations for Software Defect Prediction: Datasets, Validation Methodologies, Prediction Approaches, and Tools,” *IEEE Access*, vol. 13, pp. 866–903, 2025, doi: https://doi.org/10.1109/access.2024.3517419.

[13] P. Y. P. Chan and J. Keung, “Validating Unsupervised Machine Learning Techniques for Software Defect Prediction With Generic Metamorphic Testing,” *IEEE Access*, vol. 12, pp. 165155–165172, 2024, doi: https://doi.org/10.1109/access.2024.3494044.

[14] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” *International Conference on Software Engineering*, May 2005, doi: https://doi.org/10.1145/1062455.1062514.

[15] C. Rahmani and Deepak Khazanchi, “A Study on Defect Density of Open Source Software,” pp. 679–683, Aug. 2010, doi: https://doi.org/10.1109/icis.2010.11.

[16] D. K. Verma and S. Kumar, “Prediction of Defect Density for Open Source Software using Repository Metrics.,” *Journal of Web Engineering*, vol. 16, pp. 293–310, Jun. 2017.

[17] N. Mandhan, D. K. Verma, and S. Kumar, “Analysis of approach for predicting software defect density using static metrics,” May 2015, doi: https://doi.org/10.1109/ccaa.2015.7148499.

[18] A. Marchenko and P. Abrahamsson, “Predicting Software Defect Density: A Case Study on Automated Static Code Analysis,” *Agile Processes in Software Engineering and Extreme Programming*, pp. 137–140, Jul. 2007, doi: https://doi.org/10.1007/978-3-540-73101-6_18.

[19] D. Verma and S. Kumar, “An Improved Approach for Reduction of Defect Density Using Optimal Module Sizes,” *Advances in Software Engineering*, vol. 2014, pp. 1–7, 2014, doi: https://doi.org/10.1155/2014/803530.

[20] Z. Li, P. Liang, and B. Li, “Relating Alternate Modifications to Defect Density in Software Development,” pp. 308–310, May 2017, doi: https://doi.org/10.1109/icse-c.2017.132.

[21] M. Parastoo, R. Conradi, O. M. Killi, and H. Schwarz, “An empirical study of software reuse vs. defect-density and stability,” pp. 282–292, May 2004, doi: https://doi.org/10.5555/998675.999433.

[22] M. Sherriff, N. Nagappan, L. Williams, and M. Vouk, “Early estimation of defect density using an in-process Haskell metrics model,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–6, May 2005, doi: https://doi.org/10.1145/1082983.1083285.

[23] O. Kutlubay, B. Turhan, and A.B. Bener, “A Two-Step Model for Defect Density Estimation,” Aug. 2007, doi: https://doi.org/10.1109/euromicro.2007.13.

[24] P. Knab, M. Pinzger, and A. Bernstein, “Predicting defect densities in source code files with decision tree learners,” *Mining Software Repositories*, May 2006, doi: https://doi.org/10.1145/1137983.1138012.

[25] Y. Villuendas-Rey, A. Bou Nassif, A. Bou-Nassif, and S. Banitaan, “Upsilon-SVR Polynomial Kernel for Predicting the Defect Density in New Software Projects,” *International Conference on Machine Learning and Applications*, Dec. 2018, doi: https://doi.org/10.1109/icmla.2018.00224.

[26] C. López-Martín, Y. Villuendas-Rey, M. Azzeh, A. Bou 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, Sep. 2020, doi: https://doi.org/10.1016/j.jss.2020.110592.

[27] S. Rathaur, N. Kamath, and U. Ghanekar, “Software Defect Density Prediction based on Multiple Linear Regression,” *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, Jul. 2020, doi: https://doi.org/10.1109/icirca48905.2020.9183110.

[28] F. Alghanim, M. Azzeh, A. El-Hassan, and H. Qattous, “Software Defect Density Prediction Using Deep Learning,” *IEEE Access*, vol. 10, pp. 114629–114641, 2022, doi: https://doi.org/10.1109/access.2022.3217480.

[29] 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 / Computers and Artificial Intelligence*, vol. 32, no. 1, pp. 203–224, Mar. 2013.

[30] H. B. Yadav and D. K. Yadav, “A fuzzy logic based approach for phase-wise software defects prediction using software metrics,” *Information and Software Technology*, vol. 63, pp. 44–57, Jul. 2015, doi: https://doi.org/10.1016/j.infsof.2015.03.001.

[31] S. K. Khalsa, “A Fuzzified Approach for the Prediction of Fault Proneness and Defect Density,” Jan. 2009.

[32] [M. Azzeh, Y. Alqasrawi, and Y. Elsheikh, “A soft computing approach for software defect density prediction,” *Journal of Software: Evolution and Process*, Mar. 2023, doi: https://doi.org/10.1002/smr.2553.

**Jasmeet Kaur** did her B.Tech in Information Technology from Guru Tegh Bahadur Institute of Technology, India and her M.Tech in Information Technology from University School of Information and Communication Technology (USICT), Guru Gobind Singh Indraprastha University (GGSIPU), India. Now she is working towards her Ph.D. in Information Technology at USICT, GGSIPU. The area of her doctoral research is focused on Machine Learning based techniques for defect prediction

**Arvinder Kaur** is a Professor at the University School of Information, Communication, and Technology (USICT) and Dean of the University School of Automation & Robotics (USAR) at Guru Gobind Singh Indraprastha University (GGSIPU), Delhi. She holds a Ph.D. from GGSIPU, an M.E. in Computer Engineering, and a B.E. in Electrical Engineering from Thapar Institute of Engineering and Technology, Patiala. She has served as the Dean of USICT and is the Chairperson of the Staff Development Cell at GGSIPU. Her research interests include software engineering, artificial intelligence, and computer networks. Arvinder Kaur is also a lifetime member of ISTE and CSI.

**Kamaldeep Kaur** works as an Associate Professor at the University School of Information, Communication and Technology at Guru Gobind Singh Indraprastha University located in Delhi, India. In 2016, she graduated with her doctorate from Guru Gobind Singh Indraprastha University. Her research works focus on Neural Networks, Natural Language Processing and Software Engineering. She is a lifetime member of Indian Society of Technical Education. A number of her research papers have been published in journals listed in Web of Science and presented at IEEE conferences. She is the chairperson of IEEE Women in Engineering Chapter at University School of Information and Communication Technology, Guru Gobind Singh Indraprastha University.