FOURRAGES ISSN:0429-2766

DOI Link: https://doi.org/10.61586/bBbsj
Vol.263, Issue.8, Part.1, August 2025, PP.22-39

SMoBaT: Improving Software Defect Prediction via Balanced Ensemble Learning and SHAP Interpretability

Rohit Yadav[1*], Raghuraj Singh[2]

[1]Research Scholar, Department of Computer Science and Engineering, Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh-226021, India,
e-mail: rohitatknit@gmail.com, ORCID: 0000-0002-6792-8845
[2]Professor, Department of Computer Science and Engineering, Harcourt Butler Technical University, Kanpur, Uttar Pradesh- 208002, India,
e-mail: rrsingh@hbtu.ac.in, ORCID: 0000-0002-1718-8324
*Corresponding Author: rohitatknit@gmail.com

Received Feb 2025 Accepted July 2025 Published August 2025

Abstract

Software Defect Prediction (SDP) is crucial in improving software quality by identifying defect-prone modules early in the development lifecycle. However, class imbalance in most SDP datasets often biases machine learning models toward the majority class, reducing their ability to detect actual defects. To address this challenge, we propose SMoBaT. This hybrid framework integrates the Synthetic Minority Over-sampling Technique (SMOTE) for data balancing with Bagged Decision Trees for robust and stable prediction. SMoBaT is evaluated on two widely used benchmark datasets, KC1 and KC2, and demonstrates superior performance compared to traditional classifiers and recent state-of-the-art models. Notably, SMoBaT achieves a test accuracy of 96.77% on the KC1 dataset and 93.85% on KC2, outperforming prior methods in precision, recall, and F1-score. Additionally, we incorporate Shapley explanations (SHAP) analysis to interpret model predictions and identify the most influential software metrics contributing to defects. The results validate the effectiveness of SMoBaT in prediction accuracy and model transparency, highlighting its potential for practical deployment in software quality assurance workflows.

Keywords: Data Balancing; Ensemble Learning; Explainable Artificial Intelligence; Software Defect Prediction.

1. Introduction

Software defects, commonly referred to as bugs, are flaws or errors in a software system that can compromise its functionality, reliability, and security. Detecting and resolving these defects is crucial to maintaining software quality, especially during the maintenance phase of the Software Development Life Cycle (SDLC). However, manual defect detection is time consuming, labour-intensive, and costly, often accounting for up to 80% of total development expenses. To address this challenge, Software Defect Prediction (SDP) models have emerged as an effective means to automatically identify defect-prone modules early in the development process. By leveraging historical data and machine learning techniques, these models assist developers in prioritizing testing efforts and optimizing resource allocation [1].

Software Defect Prediction (SDP) aims to identify defect-prone software modules in advance, allowing developers to allocate testing efforts more efficiently. However, a major challenge in SDP is the significant class imbalance in most benchmark datasets, where the number of non-defective instances far outweighs the defective ones. This imbalance often causes machine learning models to become biased toward the majority class, leading to high overall accuracy but poor recall for the minority (defective) class. Consequently, many defects remain undetected, reducing the model's practical effectiveness in real-world scenarios [2].

Traditional machine learning algorithms, such as decision trees, support vector machines, and logistic regression, have been widely used in SDP. While these models are interpretable and straightforward, they often lack robustness when exposed to data distribution shifts or noisy, incomplete data. Their performance can degrade significantly under slight perturbations in the input, making them unreliable for safety-critical applications. Moreover, these models typically assume a balanced data distribution and fail to generalize well when applied to imbalanced or heterogeneous datasets commonly found in SDP [3].

Another critical yet often overlooked aspect of SDP is model interpretability. In practical settings, especially in software engineering tasks, stakeholders require accurate predictions and understandable justifications for those predictions. Despite their high accuracy, Black-box models offer little transparency into their decision-making process, limiting their adoption in industrial settings. Enhancing model interpretability helps developers understand which software metrics contribute most to defects, facilitates model debugging, and builds trust in automated decision systems [4].

In the present work, we proposed a hybrid novel approach SMoBaT, which combines the results of Synthetic Minority Oversampling Technique (SMOTE) followed by Bagged Trees (Bag) ensemble to develop the SDP. Our approach addresses the three main issues above mentioned. Data imbalance issue, accuracy issue of traditional ML, and SDP interpretability. We conducted the experiments of two object oriented based datasets (i.e., KC1 and KC2). First, we solve the problem of data imbalance utilizing SMOTE then balanced dataset is fed to the training models which includes four baseline traditional ML including decision tree (DT), naïve bayes (NB), Support Vector Machine (SVM), and K-nearest neighbor (KNN) and our proposed ensemble method Bagged Tree. For creation and the datasets were

divided into two parts. First part (75%) is used for training and remaining part (25%) is used for testing of the developed SDP model. Finally, we applied Shapely Analysis (SHAP) on our proposed SMoBaT based SDP model to interpret our model for decision making profit. We also validated our work by comparing the results of performance of our proposed model with the traditional ML approaches and existing studies. The results showed that our proposed SDP model overqualified than the state-of-the-art SDP methods. We present three main contributions in this work that are as follows:
(i) Mitigated the impact of data imbalance on accuracy of SDP by applying SMOTE
(ii) enhanced the performance of proposed SDP model than the state of the art.
(iii) applied SHAP analysis for proposed SDP model interpretation and decision making that helps researchers and practitioners.

The remainder of this paper is organized as follows: Section 2 reviews recent studies related to Software Defect Prediction (SDP). Section 3 outlines the methodology used in this research. Section 4 presents the experimental setup, implementation details, and results. Section 5 presents the threats to validity. Finally, Section 5 concludes the study.

2. Related Work

This section presents recent methods, performance metrics, and results reported in the latest studies on Software Defect Prediction (SDP). A wide range of machine learning and optimization-based approaches have been proposed to improve the accuracy and efficiency of SDP. For example, Mahmoud *et al.* [5] introduced a hybrid model combining Convolutional Neural Networks (CNN) with Antlion Optimization (ALO), where ALO was used for weight optimization. Their hybrid approach significantly outperformed the standalone CNN, GRU, BiLSTM, and Deep Forest models in terms of AUC, sensitivity, specificity, and accuracy.

Nevendra and Singh [6] proposed a metrics-based CNN (MB-CNN) that explicitly utilizes software metrics as input features. Their model achieved substantial improvements over Li's baseline CNN—31% for within-version and 28% for cross-version predictions—supported by statistical validation using Friedman and Wilcoxon tests.

Manjula and Florence [7] developed a hybrid model integrating Genetic Algorithms (GA) for feature optimization with Deep Neural Networks (DNN) enhanced by adaptive auto-encoders. Their model achieved high classification accuracy, exceeding 97% across several PROMISE datasets (KC1, CM1, PC3, and PC4).

Zaibi *et al.* [8] employed an Incremental Discriminant-based Support Vector Machine (IDSVM) for real-time defect prediction, allowing the model to evolve dynamically as new software data becomes available.

Elci and Nandagopalan [9] combined Kernel Extreme Learning Machine (KELM) with War Strategy Optimization (WSO) for hyperparameter tuning. Their model demonstrated high predictive performance, achieving 0.97 accuracy and 0.96 F1-score, showcasing the utility of kernel-based classifiers in defect prediction.

Wang *et al.* [10] proposed a Binary Gray Wolf Optimizer (bGWO) for feature selection, followed by various machine learning classifiers. The selected optimal feature subsets led to marked improvements in accuracy, precision, recall, and F1-score.

Shankar and Chaudhari [11] introduced a hybrid BAT algorithm combined with CNN, evaluated on four PROMISE datasets. Their model consistently outperformed traditional machine learning and ensemble models, achieving accuracies above 93%.

Liapis *et al.* [12] integrated active learning with ensemble classifiers using diverse query strategies (uncertainty, margin, entropy). Their method effectively reduced labeling efforts by shrinking the training set size by 75% in most cases, while maintaining strong AUC, Kappa, and MCC performance.

Sánchez-García *et al.* [13] investigated the impact of class balancing techniques across multiple classifiers. Their study confirmed that balancing classes significantly improves standard classification metrics, validated through hypothesis testing on PROMISE datasets.

Zada *et al.* [14] proposed an AI-based approach targeting IoT-based software systems by combining KELM with WSO for hyperparameter optimization. Their model achieved over 90% accuracy across various datasets, demonstrating its effectiveness in resource-constrained, real-world scenarios.

Siddika *et al.* [15] compared seventeen machine learning algorithms for SDP and reported that Gradient Boosting AdaBoost achieved the highest accuracy, followed closely by AdaBoost. While ensemble methods were employed, the study did not address data imbalance issues or provide model interpretability insights.

Kanaujiya and Verma [16] proposed an ensemble-based feature selection approach, Ensemble_ML, for SDP. Their method used Yeo-Johnson transformation and SMOTE for data balancing, followed by ensemble learning for final prediction. However, they did not examine model interpretability.

In another work, Hu *et al.* [17] introduced a federated oversampling learning framework (Fed-OLF), which merges TabDiT for data balancing and employs parameter aggregation for optimization. This framework was evaluated on NASA and PROMISE datasets, demonstrating improved performance over baseline models. However, the study lacked model interpretation and did not incorporate ensemble techniques.

Table 1 summarizes recent studies on Software Defect Prediction (SDP), highlighting data balancing techniques, ensemble methods, model interpretability, and reported limitations.

Despite these advancements, critical research gaps remain in software defect prediction. Many recent studies overlook or insufficiently address the class imbalance issue, which can significantly skew predictive performance.

Table 1. Summary of recent studies based on software defect prediction

Study	Data Balancing	Ensemble Technique	Model Interpretability	Limitation
Mahmoud et al. [5]	×	×	×	lacks balance handling and interpretability
Nevendra and Singh [6]	×	×	×	Focused only on CNN; interpretability and data imbalance not addressed
Manjula and Florence [7]	×	×	×	Uses DNN with GA but no ensemble or interpretability analysis
Zaibi et al. [8]	×	×	×	lacks ensemble methods and interpretability
Elci and Nandagopalan [9]	×	×	×	no interpretability or balancing
Wang et al. [10]	×	×	×	Effective feature selection but no ensemble or balancing strategy
Shankar and Chaudhari [11]	×	✓	×	no interpretability or class balance considered
Liapis et al. [12]	×	✓	×	lacks model interpretability
Sánchez-García et al. [13]	✓	×	×	Focused on balancing, but lacks interpretability and ensemble methods
Zada et al. [14]	×	×	×	interpretability and ensemble integration not explored
Siddika *et al.* [15]	✓	✓	×	lacks interpretability
Kanaujiya and Verma [16]	✓	✓	×	lacks interpretability
Hu *et al.* [17]	✓	×	×	interpretability and ensemble integration not explored

While techniques like SMOTE, ADASYN, or undersampling have shown effectiveness, their integration with advanced learning models, particularly ensemble approaches, remains limited.

Additionally, although ensemble methods such as bagging, boosting, and stacking have proven to enhance classification accuracy, their application in SDP is relatively sparse compared to single-model architectures. Finally, SDP model

interpretability is vital for debugging and actionable insights in software engineering, which is often neglected, with most deep learning-based solutions functioning as black boxes. Therefore, there is a growing need for balanced, interpretable, and ensemble-driven models that offer high accuracy and support transparency and generalizability across diverse software systems.

3. Methodology

In this section, we provide a detailed description of the proposed method. Fig. 1 illustrates the overall framework of the Software Defect Prediction (SDP) model, which is organized into four key phases: (i) Data Collection, (ii) Data Balancing using SMOTE, (iii) Model Training, Testing, and Performance Evaluation using the proposed SMoBaT-based SDP approach, and (iv) Interpretation of the proposed SDP model.

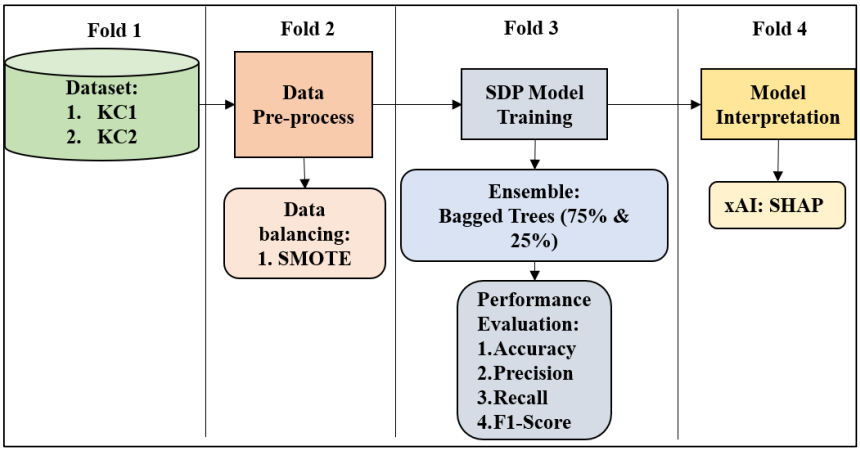

Fig. 1. Framework for proposed SDP model

A. Data Collection

This study uses two software programs (i.e., KC1 and KC2). This software is written in C++ and demonstrates the object-oriented approach used in this study. Researchers and practitioners can download this software from the link: https://github.com/ApoorvaKrisna/NASA-promise-dataset-repository. The KC1 dataset contains 2019 instances, and the KC2 dataset contains 522 cases. Each instance contains the values of 21 features related to the static code of software and a response class value with the Boolean value 'true' or 'false'. Where 'true' represents that the class is defective, and 'false' means that the class has no defects. We describe these features and the response variable in Table 2.

Table 2. Description of features present in KC1 and KC2 dataset

S.N.	Feature	Description
1	*loc*	Line count of code (McCabe)
2	*v_g_*	Cyclomatic complexity (McCabe)
3	*ev_g_*	Essential complexity
4	*iv_g_*	Design complexity
5	*n*	Total operands and operators (Halstead)
6	*v*	Volume (Halstead)
7	*l*	Program length
8	*d*	Difficulty
9	*i*	Intelligence
10	*e*	Effort
11	*b*	Estimated number of bugs
12	*t*	Time to implement (seconds)
13	*lOCode*	Lines of code
14	*lOComment*	Lines of comments
15	*lOBlank*	Blank lines
16	*lOCodeAndComment*	Code + comment lines
17	*uniq_Op*	Unique operators
18	*uniq_Opnd*	Unique operands
19	*total_Op*	Total operators
20	*total_Opnd*	Total operands
21	*branchCount*	Branch count from control flow
22	*defects*	Defect label (true/false)

B. Data Preprocessing: SMOTE

To address the class imbalance commonly observed in Software Defect Prediction (SDP) datasets, we applied the Synthetic Minority Over-sampling Technique (SMOTE) during the data preprocessing stage. SMOTE is a widely used resampling method in binary classification problems that generates synthetic samples rather than simply duplicating existing instances of the minority class. It does so by interpolating between a given minority class sample and its nearest neighbors, thus creating more diverse and generalizable examples [18].

This strategy helps reduce classifier bias toward the majority class and promotes better recall for the minority (defective) class. As a result, models trained on SMOTE-balanced datasets tend to perform more reliably on imbalanced data. The detailed procedure for generating the balanced dataset using SMOTE is outlined in Algorithm 1.

Algorithm 1: SMOTE

1. Input: Minority class samples, desired number of synthetic samples (N), number of nearest neighbors (k)

2. For each minority class sample x_i:
   - Find its k nearest neighbors from the same class.
   - Randomly select N neighbors from these k.
3. For each selected neighbor x_{nn}:
   - Compute the difference: $diff = x_{nn} - x_i$
   - Multiply the difference by a random number between 0 and 1.
   - Add this to x_i to generate a new synthetic sample: $x_{new} = x_i + rand\ (0,1) \times diff$
4. Repeat the process until the desired number of synthetic samples is created.
5. Output: Extended dataset with synthetic minority class instances.

C. SDP Model: Bagged Trees

To enhance the robustness of traditional machine learning algorithms such as Decision Trees (DT), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Naïve Bayes (NB), we employed an ensemble learning approach in this study. Specifically, we utilized Bagged Trees, also known as Bootstrap Aggregated Decision Trees [19].

Bagged Trees improve base learners' stability and predictive performance, particularly decision trees, by reducing variance. The method involves generating multiple bootstrap samples (random samples with replacement) from the original training dataset. Each sample is used to train an independent decision tree, and the final prediction is obtained by aggregating the outputs of all trees (e.g., through majority voting for classification tasks) as illustrated in Fig. 2. Our proposed SMoBaT-based Software Defect Prediction (SDP) model comes under a binary classification problem, where each software defect prediction dataset instance's response variable is either 'true' or 'false', representing *a defective module* or *a non-defective module, respectively*. Given this setup, Bagged Trees are a suitable choice due to their effectiveness in handling binary classification tasks and their robustness in class imbalance and noisy data.

We split each dataset (i.e., KC1 and KC2) into 75% for training and 25% for testing to train and evaluate the model. This split ensures that the model is trained on a substantial portion of the data while maintaining a separate and unseen set for performance evaluation.

D. Model Interpretability: SHAP Analysis

In the final phase of our SDP model construction, we applied SHAP to interpret the predictions made by the proposed SMoBaT-based model. SHAP provides a unified and theoretically grounded framework for model interpretability by assigning each feature a contribution score based on its influence on the model's output [20].

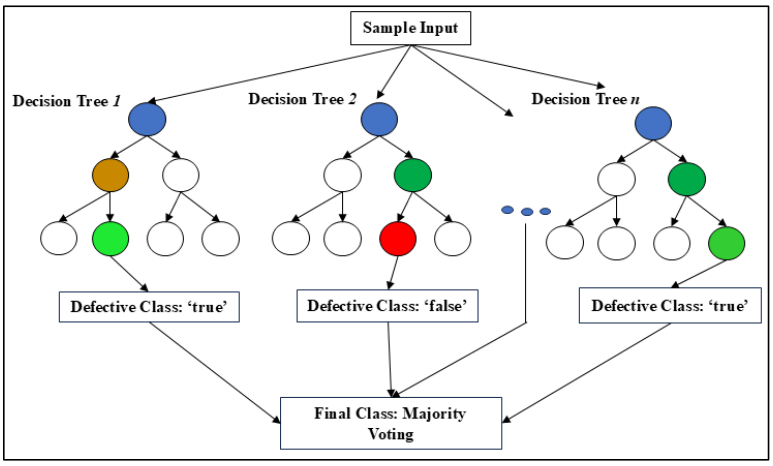

Fig. 2. Framework of Bagged trees

In the SDP context, SHAP values help quantify the contribution of individual software metrics such as lines of code, cyclomatic complexity, and coupling to the classification outcome (defective or non-defective). As our proposed SMoBaT-based SDP is a binary classification problem, SHAP operates class-wise, explaining how feature values affect predictions for each class separately. For instance, in a binary classification scenario, a positive SHAP value for a feature in the *defective* class indicates that the feature increases the likelihood of a module being classified as faulty. Conversely, a negative SHAP value for a feature in the *non-defective* class suggests that lower values of that feature contribute to predicting the module as clean. This interpretability step provides deeper insight into the SDP model's decision-making process and helps identify key metrics that influence software defect predictions.

4. Experimental Setup

This section outlines the experimental design and evaluation details for the proposed Software Defect Prediction (SDP) model.

A. Research Questions

To address the existing research gaps, we aim to evaluate the performance and interpretability of our proposed model by answering the following research questions:
RQ1: Does the proposed approach outperform traditional machine learning techniques commonly used in SDP?
RQ2: Does the proposed SDP model achieve better results than recent state-of-the-art approaches?

RQ3: What insights can be gained by incorporating SHAP value analysis into the SDP model?

The objective of **RQ1** is to identify whether the proposed model (SMoBaT) delivers superior performance compared to conventional ML models. **RQ2** evaluates the effectiveness of our model compared to existing advanced techniques reported in the literature. Finally, **RQ3** focuses on model interpretability, assessing how SHAP value analysis can enhance understanding and support more informed decision-making in the context of software defect prediction.

B. Parameter Settings

This section describes the various parameters used to implement the SMoBaT. First, we applied SMOTE to our dataset KC1 with the following parameter values. In this study, we implemented SMOTE using the WEKA tool. We used six parameters to produce the results of SMOTE (i.e., *ClassValue, debug, doNotCheckCapabilities, nearestNeighbors, percentage, and random seed*. The *ClassValue* indicates which class (i.e., defective class: true or false). If set to true, the *debug* is a boolean type parameter that enables additional information on the console. *DoNotCheckCapabilities* is also a boolean variable. If true, then capabilities are not checked before applying the SMOTE technique to the dataset. The values present in the dataset are compatible or not. The *nearest neighbors* determine the number of neighbors considered while creating the samples. The *percentage* and *random seed* determine the portion of the produced synthetic data and the number of seeds required. In this study, we applied the SMOTE. We set the following parameter values: *ClassValue* to zero, *debug* to false, *doNotCheckCapabilities* to false, *nearestneighbors* are set to 5, while *percentage* and *random seed* values are 100 and one, respectively.

To implement the Bagged Trees, we utilized the MATLAB tool. We considered the following parameters: the *bag* method is selected, the decision tree is chosen as *learner type*, with a *maximum number of splits* of 2108 for KC1 and 521 for KC2, based on the instances both datasets have. And we selected all *features* in the training and testing of SDP construction.

C. Performance Parameters

We used four well-known parameters: accuracy, precision, recall, and F1-score. Also represents the comparison of our work with traditional ML by the ROC curve. The confusion matrix of the SDP built can evaluate all four metrics. Table 3 shows the structure of the confusion matrix.

A confusion matrix in a binary classification problem contains four cells: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TP shows the number of instances correctly predicted and having the actual class value. TN indicates the number of cases correctly predicted as negative classes and

that were negative. Meanwhile, FP and FN are instances wrongly predicted by the model as true and false, respectively.

Table 3. confusion matrix

	Positive	Negative
True	TP	TN
False	FP	FN

The accuracy, precision, recall, and F1-score can be determined by the values of a confusion matrix presented in Table 4. The formula for accuracy, precision, recall, and F1-score are provided in equation (1), equation (2), equation (3), equation (4), respectively.

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

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

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

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

The ROC (Receiver Operating Characteristic) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier. It shows the trade-off between the True Positive Rate (TPR) on the Y-axis and the False Positive Rate (FPR) on the X-axis across different classification thresholds. A curve closer to the top-left corner indicates better performance, while the Area Under the Curve (AUC) quantifies the overall model effectiveness.

5. Results and Analysis

We provide results of our experiments answering the research questions.

Answer to RQ.1 *Does the proposed approach outperform traditional machine learning techniques commonly used in SDP?* We calculated the accuracy, precision, recall, and F1-score of our proposed SDP and these values for four traditional ML-based SDPs on the datasets KC1 and KC2, respectively. The values of these metrics are present in Tables 4 and 5, respectively.

We evaluated the performance of various SDP models using the KC1 dataset, which consists of 1,582 observations, 21 software metrics as predictors (e.g., LOC, cyclomatic complexity, Halstead metrics), and a binary response variable representing defect presence (true or false). Table 4 presents the comparative performance of five models: Decision Tree (DT), Naive Bayes (NB), Support

Vector Machine (SVM), K-Nearest Neighbors (KNN), and the proposed SMoBaT (SMOTE + Bagged Trees) approach. Across validation (75%) and test (25%) splits, SMoBaT significantly outperformed the baseline models in all evaluation metrics, achieving a test accuracy of 96.77% and an F1-score of 96.70%. In contrast, traditional classifiers like SVM and KNN achieved lower test accuracies (85.58% and 85.01%, respectively) and comparatively moderate F1-scores (80.62% and 80.96%). Results indicate that SMoBaT addresses class imbalance effectively through SMOTE and enhances generalization and stability via ensemble bagging. The consistent gains in precision, recall, and F1-score confirm the robustness and reliability of the proposed model in accurately predicting software defects.

We again applied the proposed SMoBaT model on the KC2 dataset, which contains 392 observations and the same binary class structure (true and false) for defect prediction. Table 5 compares the performance of standard classifiers (DT, NB, SVM, KNN) against SMoBaT using both validation (75%) and test (25%) data splits.

Table 4. Values of performance parameters of SDP models applied with KC1 dataset.

Method Type	Validation (75% dataset)				Test (25% dataset)			
	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
DT	81.54	79.91	81.54	80.62	83.11	80.44	83.11	81.40
NB	82.36	81.86	82.36	82.10	82.73	81.77	82.73	82.21
SVM	85.40	83.12	85.40	80.58	85.58	83.84	85.58	80.62
KNN	84.96	81.54	84.96	81.41	85.01	81.40	85.01	80.96
SMoBaT	93.36	93.09	93.36	93.09	96.77	96.73	96.77	96.70

The results show that SMoBaT outperforms all baseline models, achieving a test accuracy of 93.85% and an F1-score of 93.65%, significantly higher than other models. For instance, Naive Bayes (NB) achieved a test accuracy of 85.38% and an F1-score of 83.50%, while SVM and KNN achieved 83.85% and 80.77%, respectively. Traditional models showed a noticeable drop in performance on the test set, highlighting potential overfitting or sensitivity to data imbalance. In contrast, SMoBaT maintained high recall and precision values, demonstrating its robustness and reliability.

Table 5. values of performance parameters of SDP models applied with KC2 dataset.

Method Type	Validation (75% dataset)				Test (25% dataset)			
	Accuracy	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score
DT	82.14	81.66	82.14	81.88	80.00	81.15	80.00	80.49
NB	83.16	81.69	83.16	81.99	85.38	84.62	85.38	83.50
SVM	84.69	83.84	84.69	82.30	83.85	83.30	83.85	80.75
KNN	83.42	81.72	83.42	81.39	80.77	80.51	80.77	80.63
SMoBaT	88.27	87.69	88.27	87.62	93.85	93.74	93.85	93.65

To strengthen our results, we also present a ROC curve comparison with traditional SDP models based on Decision Tree (DT), Naïve Bayes (NB), Support Vector Machine (SVM), and k-Nearest Neighbour (KNN), as shown in Fig. 3. Fig. 3(a) compares the performance of traditional ML-based SDP models and our proposed SMoBaT model on the KC1 dataset for the 'false' class (non-defective instances). SMoBaT outperforms the baseline models, as indicated by an ROC curve enclosing a larger area under the curve. Similarly, Fig. 3(b) shows the ROC curve comparison on the KC2 dataset for the 'false' class, confirming that our proposed method consistently outperforms traditional approaches across different datasets.

Answer to RQ.2 *Does the proposed SDP model achieve better results than recent state-of-the-art approaches?* To address this question, we compared the performance of our proposed Software Defect Prediction (SDP) model, SMoBaT (SMOTE + Bagged Trees), with the state-of-the-art approach (Shankar and Chaudhari [11] and Al-Fraihat et al. [21]). Shankar and Chaudhari [11] presented a hybrid model, which combines the BAT algorithm and Convolutional Neural Networks (CNN), and achieved an accuracy of 92% on the KC1 dataset. In contrast, our proposed SMoBaT model demonstrated superior performance during the training and testing.

In the training phase, SMoBaT achieved an accuracy of 93.36%, precision of 93.09%, recall of 93.36%, and F1-score of 93.09%. More importantly, in the testing phase, which better reflects the model's generalization capability, our approach achieved significantly higher metrics: 96.77% accuracy, 96.73% precision, 96.77% recall, and 96.70% F1-score. These results indicate that SMoBaT not only outperforms the existing BAT-CNN hybrid model in terms of all four-evaluation metrics but also offers a more balanced and robust performance, especially when applied to unseen data.

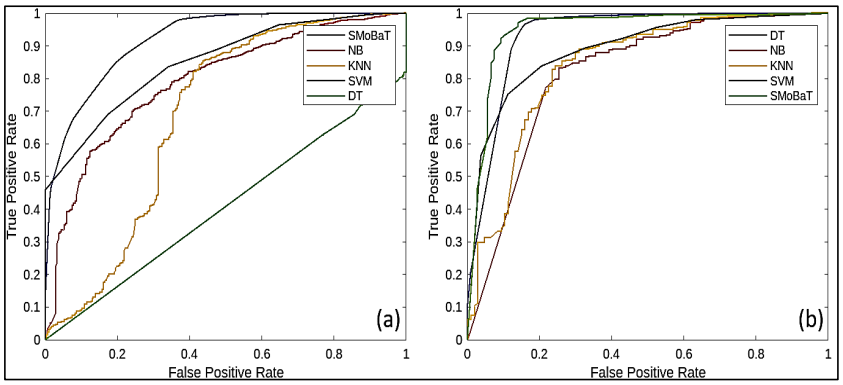

Fig. 3 ROC curve Comparison of DT, NB, SVM, and SMoBaT based SDP developed using (a) KC1 dataset and (b) KC2 dataset

Answer to RQ.3 *What insights can be gained by incorporating SHAP value analysis into the SDP model?* To answer this, we performed SHAP analysis on the KC1 and KC2 datasets to address this question using our proposed SMoBaT model (Fig. 4). We presented the SHAP swarm plots to visualise the results for both class labels, 'false' (non-defective modules) and 'true' (defective modules). In these plots, the color intensity represents the SHAP value magnitude for each feature: darker colors indicate higher SHAP values, signifying a more substantial impact of that feature on the model's prediction. A positive SHAP value with high intensity suggests that higher values of that feature contribute more towards predicting the module as defective.

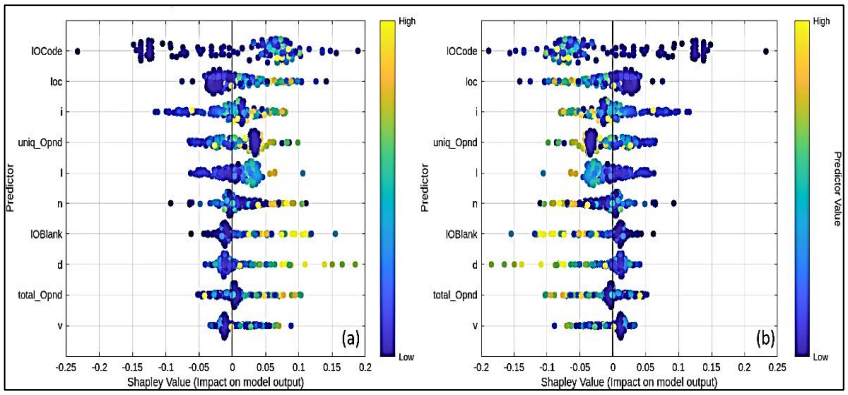

Fig. 4 Swarm plot of the SMoBaT-based SDP model showing SHAP values of features contributing to the prediction of (a) defective and (b) non-defective classes for KC1

For the KC1 dataset, Fig. 4(a) illustrates the SHAP swarm chart for the 'false' class, highlighting the top 10 contributing features: *IOCode, loc, I, uniq_Opnd, l, n, IOBlank, d, total_Opnd,* and *v*. The analysis reveals that for accurate prediction of non-defective instances (class = false), lower values of most of these features are more favorable. Conversely, Fig. 4(b), which presents the SHAP analysis for the 'true' class, shows that higher values of these same features improve the prediction of defective modules. For instance, a higher *loc* value has a positive SHAP impact in identifying a module as defective, suggesting that large code size may indicate defects.

Similarly, for the KC2 dataset, Fig. 5(a) presents the SHAP values for the 'false' class, with the top 10 features identified as: *total_Opnd, I, ev_g_, loc, uniq_Opnd, n, d, e, v_g_*, and *l*. The findings show that lower values of features like *total_Opnd, ev_g_, uniq_Opnd, n, d, e*, and *v_g_* support better classification of non-defective modules, while features such as *I, loc*, and *l* have a more positive impact when their values are higher. Conversely, Fig. 5(b) shows that higher values of most features favor accurately classifying defective modules.

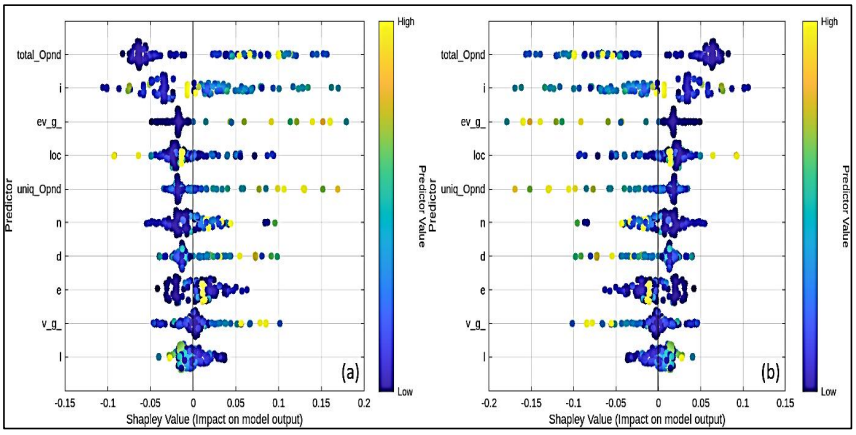

Fig. 5 Swarm plot of the SMoBaT-based SDP model showing SHAP values of features contributing to the prediction of (a) defective and (b) non-defective classes for KC2

Overall, the inclusion of SHAP analysis enhances the interpretability of the SDP model and helps identify which features and value ranges are most influential in predicting defective versus non-defective software modules. This insight supports better understanding, debugging, and trust in model decisions.

6. Threats to Validity

Despite the promising results of the proposed SMoBaT based SDP model, we acknowledge specific threats to validity. Internal validity can be affected by the choice of parameters and dataset-specific characteristics, which could affect model performance. External validity could be affected by publicly available datasets (e.g., KC1, KC2), which may not generalize to all software projects. Construct validity could be affected by the selection of software metrics and the binary classification formulation, which may oversimplify real-world defect scenarios. Finally, conclusion validity depends on the correctness of performance metrics and the statistical significance of observed improvements. We employed Standard preprocessing, consistent train-test splits, and well-established evaluation measures throughout the experiments to mitigate these threats.

7. Conclusion

This study introduced SMoBaT, a novel and effective hybrid model that integrates SMOTE-based oversampling with Bagged Decision Trees to address class imbalance and improve prediction accuracy in Software Defect Prediction (SDP) tasks. The experimental results on two benchmark datasets, KC1 and KC2, demonstrate that SMoBaT consistently outperforms traditional classifiers and state-of-the-art approaches (e.g., Shankar and Chaudhari [11] and Shankar and Chaudhari [11] and Al-Fraihat et al. [21]) across all key performance metrics. Furthermore, the inclusion of SHAP value analysis enhances the interpretability of the model by identifying the most influential software metrics contributing to defect prediction. Overall, SMoBaT offers robust predictive performance and supports transparent, explainable decision-making, making it a promising tool for real-world defect management in software engineering. In the future, we plan to extend this work by evaluating the model on larger and more diverse industrial datasets, incorporating cross-project defect prediction scenarios, and exploring deep ensemble architectures to enhance accuracy and generalizability further.

Acknowledgement

The authors sincerely thank Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India, for providing the opportunity to conduct this research.

References

1. Fenton, N. E., & Neil, M. (2002). A critique of software defect prediction models. IEEE Transactions on software engineering, 25(5), 675-689.
2. L'heureux, A., Grolinger, K., Elyamany, H. F., & Capretz, M. A. (2017). Machine learning with big data: Challenges and approaches. Ieee Access, 5, 7776-7797.
3. Kaur, A., & Kaur, K. (2015). An empirical study of robustness and stability of machine learning classifiers in software defect prediction. In Advances in intelligent informatics (pp. 383-397). Cham: Springer International Publishing.
4. Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., & Ziviani, N. (2020). Understanding machine learning software defect predictions. Automated Software Engineering, 27(3), 369-392.
5. Mahmoud, A.N., Santos, V., Freire, M.M. et al. Enhancing software defect prediction with a hybrid convolutional neural network and antlion optimization model. Cluster Comput 28, 405 (2025). https://doi.org/10.1007/s10586-024-05010-4
6. Nevendra, M., Singh, P. Defect count prediction via metric-based convolutional neural network. Neural Comput & Applic 33, 15319–15344 (2021). https://doi.org/10.1007/s00521-021-06158-5
7. Manjula, C., Florence, L. Deep neural network based hybrid approach for software defect prediction using software metrics. Cluster Comput 22 (Suppl 4), 9847–9863 (2019). https://doi.org/10.1007/s10586-018-1696-z
8. Zaibi, D., Salhi, M., Tbarki, K., & Ksantini, R. (2025). An incremental software defect detection model based on support vector machine. Engineering Computations, 42(1), 76-95.
9. Elci, J.B., Nandagopalan, S. WSO-KELM: War Strategy Optimization-Based Kernel Extreme Learning Machine for Automatic Software Fault Prediction Model. J. Inst. Eng. India Ser. B 106, 145–163 (2025). https://doi.org/10.1007/s40031-024-01083-2
10. Wang, H., Arasteh, B., Arasteh, K., Gharehchopogh, F. S., & Rouhi, A. (2024). A software defect prediction method using binary gray wolf optimizer and machine learning algorithms. Computers and Electrical Engineering, 118, 109336.
11. Shankar, S. P., & Chaudhari, S. S. (2024). Analysis of Bio Inspired Based Hybrid Learning Model for Software Defect Prediction. SN Computer Science, 5(7), 825.
12. Liapis, C. M., Karanikola, A., & Kotsiantis, S. (2024). Data-Efficient software defect prediction: a comparative analysis of active learning-enhanced models and voting ensembles. Information sciences, 676, 120786.
13. Sánchez-García, Á. J., Limón, X., Domínguez-Isidro, S., Olvera-Villeda, D. J., & Pérez-Arriaga, J. C. (2024). Class Balancing Approaches to Improve for Software Defect Prediction Estimations: A Comparative Study. Programming and Computer Software, 50(8), 621-647.
14. Zada, I., Alshammari, A., Mazhar, A. A., Aldaeej, A., Qasem, S. N., Amjad, K., & Alkhateeb, J. H. (2024). Enhancing IOT based software defect prediction in analytical data management using war strategy optimization and Kernel ELM. Wireless Networks, 30(9), 7207-7225.
15. Siddika, A., Begum, M., Al Farid, F., Uddin, J., & Karim, H. A. (2025). Enhancing Software Defect Prediction Using Ensemble Techniques and Diverse Machine Learning Paradigms. *Eng*, *6*(7), 161.
16. Kanaujiya, G. K., & Verma, P. (2025). Ensemble-based feature selection and machine learning models for software defect prediction. *Knowledge and Information Systems*, 1-29.

17. Hu, X., Zheng, M., Zhu, R., Zhang, X., & Jin, Z. (2025). Fed-OLF: federated oversampling learning framework for imbalanced software defect prediction under privacy protection. *IEEE Transactions on Reliability*.
18. Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139, 106662.
19. Suresh Kumar, P., Behera, H. S., Nayak, J., & Naik, B. (2021). Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature. Innovations in Systems and Software Engineering, 17(4), 355-379.
20. Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., & Ziviani, N. (2020). Understanding machine learning software defect predictions. Automated Software Engineering, 27(3), 369-392.
21. Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A. R., Alshishani, H., & Algarni, A. (2024). Hyperparameter optimization for software bug prediction using ensemble learning. *IEEE Access*, *12*, 51869-51878.

***Rohit Yadav** is currently pursuing his Ph.D. at Dr. A.P.J. Abdul Kalam Technical University, Lucknow, on object-oriented software quality and software maintainability prediction. He has received his M.Tech. Degree in Computer Science and Engineering branch from Kamla Nehru Institute of Technology, Sultanpur, U.P., India in 2015.

Dr. Raghuraj Singh is currently working as a Professor of Computer Science & Engineering and Dean, Research and Development at Harcourt Butler Technical University, Kanpur (India). He has more than 34 years of experience in teaching & research. He has successfully discharged various administrative responsibilities like Director, KNIT Sultanpur, Nodal Officer, IIIT Lucknow, Director (Mentor) of Rajkiya Engineering College, Ambedkar Nagar and Sonbhadra, Nodal Officer of Rajkiya Engineering College, Bijnor, and Dean, Planning & Resource Generation, Dean, Continuing Education & Internal Quality Assurance, Head of Department, Registrar, Controller of Examination, Chief Proctor, Asstt. Dean etc. at HBTU, Kanpur. Prof. Singh has handle 08 research and consultancy projects funded by the AICTE/UGC New Delhi and various other funding agencies. He is a member of professional bodies like IETE, IE (India), CSI, ACM, ISTE, IACSIT, Singapore etc. and reviewer & member of editorial board of many National/International journals/conferences and books. He has published more than 180 research papers in National and International journals/conferences, 06 Books/Book Chapters, supervised 19 Ph.D. theses, 29 M. Tech. dissertations, and more than 90 B. Tech./MCA projects.