International Journal of Computational and Electronic Aspects in Engineering
Volume 6, Issue 3, September 2025, pp 202 - 217
https://doi.org/10.26706/ijceae.6.3.20250811

Enhanced Software Defect Prediction Using Deep Ensemble Learning for Proactive Quality Assurance

Iman Ismail Akkar [1,*], Mohammed Shabat Abdul[2], Alaa Jamal Jabbar[3]

[1] College of Medicine, Sumer University, Iraq
[2] Director of Citizens Affairs Division, Sumer University, Iraq
[3] Department of Computer Science, Directorate General of Dhi Qar Education, Iraq

emaneesmail30@gmail.com [1], mhammad.mhsh@gmail.com[2], rdaalzydy00@gmail.com[3]

*Correspondence: emaneesmail30@gmail.com

Article – Peer Reviewed
Received: 20 July 2025
Accepted: 26 August 2025
Published: 8 September 2025

Cite this article: Iman Ismail Akkar, Mohammed Shabat Abdul, Alaa Jamal Jabbar, "Enhanced Software Defect Prediction Using Deep Ensemble Learning for Proactive Quality Assurance", *International Journal of Computational and Electronic Aspects in Engineering*, RAME Publishers, vol. 6, issue 3, pp. 202-217, 2025.
https://doi.org/10.26706/ijceae.6.3.20250811

Abstract: SDP is the term that describes the most significant step in the process of identifying fault-prone components provided in the case of the software development life cycle. Its main agenda is not only to enhance software quality but also to cut down maintenance cost in the long run. Although the sheer amount of research dedicated to SDP is quite high, the current models still have numerous limitations in its practice such as high false positive rates and the serious imbalance of the number of defective and substitute modules. Natural limitations negatively affect the learning capabilities of predictive models and lower the accuracy and quality of the debugging process as a whole.

To solve these nagging issues, the current study has proposed a robust and strong SDP model that builds on a mixture of sophisticated computing techniques, in particular, those that are eventualities of making precise forecasts and model generalization. The method stated here starts with a thorough preprocessing bezel on the information, during which the information will be normalized in terms of features, so that all elements are on a similar scale. The move serves to reduce the impact of outliers and noisy data hence improving the quality of training dataset.

After preprocessing, a class imbalance will be solved with the help of Minority Oversampling by Synthetic Data (MOSD) technique. This can be done by creating synthetic data of the minority class to get a more balanced distribution of defective and good examples which is essential toward good training of classifiers. After that, an Adaptive Sequential K-Best (ASKB) feature selection algorithm is applied to highlight the most considerable and informative features. This approach analytically discerns the significance of any attribute in a dynamic fashion; thus decreasing the dimension of the dataset with negligible loss of crucial information on predictive variables. The slimmed down feature set will also help in building a model that can be interpreted better and is also computationally economical.

As a classification algorithm a Weighted Random Forest (WRF) is used in the case of the classification task. This extension of the conventional Random Forest incorporates instance-based weighting where the model is made to give more weight to developing accurate labels of instances of the minority class. The WRF classifier, in turn, improves the overall performance in prediction and lessens the bias in relation to the dominant category. The empirical assessment of the offered structure proves its capabilities to be highly advanced in comparison to the existing models. The method produced outstanding classification accuracy measures that were 99.11999, 99.43111, 99.12199 and 99.33333 in terms of accuracy rate, precision, recall and F1- score respectively. The outcomes can substantiate the usefulness of the combined methodology on enhancing defect forecasting abilities. Moreover, the suggested model has practical implications that can be employed in proactive software quality assurance, thus more reliable and cost-effective software engineering processes can be conducted.

Keywords: Class imbalance , Defect predication in software , Minority oversampling , Data preprocessing , Synthetic data ,Adaptive UIS K-best ,Weighted random forest, Feature selection , Predictive modelling.

1. Introduction

Defects in software are a long standing, vicious issue during the software development life cycle (SDLC). The software industry is estimated to lose nearly 60 billion dollars every year because of defects in software whose failure is estimated at 30-40 percent [1]. In addition to this, it has been observed that the cost of defect correctibility at design time is significantly, i.e. five to ten times cheaper, than at the market phase [2]. Such data makes the necessity of the proper and accurate Software Defect Prediction (SDP) mechanisms which may be employed during early stages of development even more crucial in terms of maintaining control over expenditures and decreasing the quality of the software.

The quality and reliability of any software systems must be given utmost importance and especially in sensitive fields like the telecommunications, healthcare and financial fields. In these fields, small flaws in the software may not only result in considerable financial losses but also in the safety hazards of serious nature and loss of reputation [3]. The challenge for software developers today is the continuous integration / continuous deployment (CI/CD) process and the growth of agile practices, this, in turn, makes it increasingly important to know in advance what type of software defect to expect, and when. Predictive analytics has also become another crucial strategy to determine the elements that may be faulty at the start of the development uprising. The impact of early detection is that it will allow the developers to deploy their resources better and direct their efforts to risky aspects or modules. The focus on those areas with the most significant risks of failing will allow teams to reduce the risks of the issues before they grow into serious ones [4]. Therefore, incorporation of high-quality defect prediction tools has now become pivotal in the provision of functional software systems that are not only user friendly, but also adhere to high standards imposed by regulatory bodies. The tools used add value to the development process as a whole by addressing software reliability, less time in market, and tracking safety and quality issues by standards that could be specific to an industry.

In the past, there has been much reliance on manual, labor-intensive behaviors in Software Defect Prediction (SDP) which includes code reviews, examine of the code, and numerous others testing actions [5]. Although they formed the basis of conventional quality assurance, these approaches were subjective in that they were highly vulnerable to human error and inconsistency. Moreover, the process behind decisions was more prone to use the heuristic way of reasoning and historical intuition that were poorly founded statistically. Consequently, forecasts were often not accurate causing ineffective utilization of resources and soaring projects' expenditures [6]. Notwithstanding their role that has been previously observed in software development, these conventional solutions have several limitations especially when it comes to scalability, adaptability, and responsiveness. In modern software development, which is under pressure to work on shorter timescales and with increasingly rigorous assurance requirements, they are less well adapted to meet those demands. "The Artificial Intelligence (AI) [7] and Machine Learning (ML) [8] became revolutionary solutions used in the recent years to predict software defects".

These technologies provide the ability to analyse very substantial data on historic and real-time data associated with software, discern underlying trends, and offer predictions that are much more accurate than what might be done manually in most cases. ML models especially through repeated learning on the MDR datasets constantly master the idea of improving predictive capacity as more data based on defects accrues with time. Use of AI and ML in the SDP process facilitated not only in improvement of the accuracy of defect detection but also the time and efforts expended by respective developers. Such a move allows the development teams to maintain a high-level process and focus their attention on more tactical issues and innovative approaches, as well as on strategic decision-making, since it is finally being relieved of the monotonous processes that were likely unreliable and work-intensive.

Although all the above have been developed, there are some challenges that the SDP based on AI- and ML also has to grapple with. One of these difficulties is the problem of the class imbalance because defective values are greatly underrepresented compared to the non-defective ones. This skewness brings the learning models towards the majority category and hence a reduced sensitivity to the actual maladies [9]. Further, the issue of inappropriate feature selection is in most of the state of the art models, since most of them contain irrelevant or redundant features that fail to capture the dynamic nature of software systems. In addition to this, the failure to interpret and the non-transparency, popularly referred to as the black box composition of the field has made it hard to adopt such models by stakeholders who must possess certainty and reasonable predictability of their predictions [10].

All these challenges must have to be tugged to get strong, concrete and explainable SDP structures which can be effectively incorporated into the contemporary software developing methods.

The remaining paper is organized in the following way: the 2 section presents the literature relevant to software defect prediction techniques. It describes the proposed approach in Section 3, that is composed of Minority Oversampling by Synthetic Data (MOSD), Adaptive Sequential K-Best (ASKB) feature selection, and Weighted Random Forest (WRF) classifier. Section 4: Here, the authors presented experimental outcome as well as the performance of proposed technique compared to the performance of other techniques. Finally, the paper is done with Section 5; they provide an overview, describe the limitations and future investigations.

2- Literature Review

Recent innovations in the Artificial Intelligence (AI) and Machine Learning (ML) world have made a considerable impact on the Software Defect Prediction (SDP) field. Different researchers have presented new ways of improving the quality, reliability, and fault identification of software to be used in different application areas.

In [11], the authors have developed an artificial intelligence (AI) based method to automatize the process of generation of test data in order to enhance the quality and reliability of the FinTech software applications. Although the approach was proven to be of use in controlled test settings, the use of the approach on complex or large software systems was quite limited. It is therefore limited in terms of it application within wider software engineering environments due to the scalability issue. Wang et al. [12] also elaborated on how AI can be applied to software testing by publishing a wide-scale survey of how software testing methods using large language models (LLMs) take shape in today. In their own review, they hierarchized the potential in LLMs to automatically produce test cases as something that holds a lot of potential and can be very useful in automating and simplifying the software verification process. These promising capabilities notwithstanding, the authors also stressed out the main limitations of such systems, the most prominent of which includes the problem of interpretability and the computational resource consumption. Such limitations impede the implementation of LLM-based testing solutions, especially in low-resource settings where infrastructure and hardware resources are limited to run such powerful classes of models.

Chen and Babar [13] were concerned with security challenges facing machine learning based software systems and propounded various counter measures that could be used to reduce such risks. Although their work is very relevant to the field of intelligent systems, and the increased concern over software security is referenced in this realm, the given work does not accurately address the field of software defect prediction (SDP). In the same manner, Kedi et al. [14] established a machine learning framework that aimed at assisting decision-making in a small to medium software enterprise. Nevertheless, they had a limited scope in evaluating quality assurance practices and concluded the study without doing a wider evaluation of software reliability or systematic fault identification, which reduces its relevance to SDP. Khan and Masum [15], used predictive analytics and machine learning in combating real-time defect detection in agile development settings. Even though their approach increased the process of testing, it cannot provide a complete solution to forecast defects within large or immensely complex software systems. In its turn, Feng et al. [16] suggested a sophisticated under sampling technique that is aimed at enhancing the ability of handling the task of imbalanced classes by adjusting the termination condition during the training. Although they are effective when used on a moderately imbalanced dataset, it does not perform optimally when exposed to datasets that are highly imbalanced and characterized by high variations, especially when used in the process of detecting examples in the minority class.

To varying degrees, these studies highlight the continued existence of a need to develop more general-purpose, scalable, explainable defect prediction models that perform well on a variety of software systems and distributions of data.

"A name suggested by Elci and Nandagopalan [17] is Sparrow Search-based Weighted Dual Recurrent Network (SS-WDRN) to predict the software faults". The approach enhanced the defect detection accuracy, however, it was not robust, noisy and had missing data problems. Based on this, Samal and Kumar [18] proposed a hybrid model, which includes Auto-Regressive Integrated Moving Average (ARIMA) and Artificial Neural Networks (ANNs), as a way of improving the prediction of software reliability. But their model did not perform well in real-time defect prediction scenarios especially in comparison to time-series specific models.

The next noteworthy use-case of transfer learning was given by Liao et al. [19], using transfer learning to make software defect prediction models applicable across various software projects through the labeling of open-source bug

reports. Although this approach helped in the cross-project learning phenomenon, it was not quite effective because of the low ability to extract features that were domain specific hence diminishing its entity in prediction.

Concerning advanced manufacturing, another example of framework upliftment of defect prediction to laser powder bed fusion (L-PBF) mechanisms was suggested by Oster et al. [20]. They introduced the concept of deep learning as a model of the defect prediction mechanism with in-situ thermography methods. Despite the good outcomes of this approach when applied under industrial contexts, the limitation on the extent to which this approach can be applied in a standard software development environment is its domain specificity. Morovati et al. [21] analyzed the possibilities of debugging the systems which were empowered by machine learning. Their piece of work gave meaningful amounts of information about the kind of bugs particular to such systems; nonetheless, it did not present a generalized or ramp-up system that can fit in larger and more extensive software project fault predictions.

The prediction of defects in semiconductor manufacturing by applying Graph Neural Networks (GNNs) was suggested by Rahman et al. [22]. The method recorded commendable success towards the optimization of the hardware related defect identification. However, its usage is mostly restricted to hardware applications leading to the fact that it may not be appropriate to use it in a general software development projects. On the whole, these studies add to the field of study, but collectively points to the lack of universally adaptable, scalable and domain independent defect prediction models in large scale software systems.

Fu et al. [23] proposed an empirical system that predicts and classifies vulnerability and categorizes vulnerability called AiBugHunter that is tailored toward discovering vulnerability and vulnerability classification in a software system. The tool has very good results in security flaws detection and relatively poor in general-purpose software defect prediction with an average performance in the more general categories.

Al Dallal et al. [24] conducted another experiment that aimed at analyzing the significance of move method refactoring on the quality of object-oriented software systems with the help of analysis based on machine learning. They showed that software design and maintainability may improve as a result of refactoring. The refactoring process alone, however, directly contributes to the occurrence of software defects nor does it predict when defects will occur hence it has limited application in proactive frameworks of defect detection.

Finally, Giray et al. [25] examined how to use deep learning to predict software defects. As they found out, the accuracy of the predictions can be enhanced enormously when deep learning models are used. However, this strategy is very reliant on large, correctly labelled datasets and huge amounts of computing power. It might not therefore be the right choice on smaller scale projects or low data ambifiers hence limiting its implementation in data scanty situations. Whilst illustrating promising progress of such studies in their specifications, these publications further illustrate the continued work into building universally-successful, scaleable, resource-efficient software defect prediction algorithms, a notion that has, once again, grown to be a pressing need.

To conclude, scalability issues, interpretability, class imbalance, or domain-specific constraints continue to prevail in spite of various studies aiding the development of the field of software quality assurance and prediction of its defects. These difficulties indicate the necessity in well-based, generic, and resource-effective SDP frameworks.

3. Proposed methodology

Figure 1 shows the diagram of the proposed system structure. The first process is data cleaning, which is a very significant pre-processing procedure to improve the quality and homogeneity of the data. It is a process that involves feature scaling whereby it scales the range of the independent variable so that every feature equally participates in training the model. Other statistical measures like Z-score analysis used in the detection of outliers also enable the elimination of such abnormal data values that would otherwise corrupt the predictivity of the model. In order to address the missing data problem, strategies of imputation are used. They include mean and mode imputation on numerical and categorical data respectively and K-Nearest Neighbours (KNN), where similarity in missing values is estimated as a dependent variable of neighbourhood similarity [26-28]. Such pre-processing techniques, in general, produce a high quality data set, which vies directly in better and more obviously correct training of the model in the further processing.

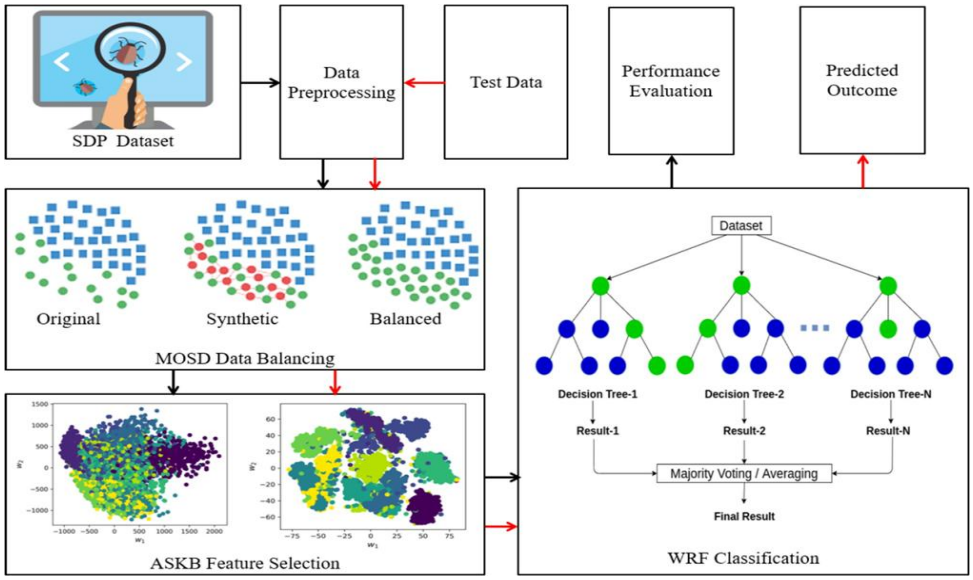

Figure 1. Proposed system model

The second step is the one to come after pre-processing data such that, it has overcome the problem of imbalance of the classes which is very common in software defect prediction problems. Using Minority Oversampling by Synthetic Data (MOSD) helps to resolve this task. As a result of the process, artificial instances of the minority class would be created since some techniques would incorporate the Synthetic Minority Over-sampling Technique (SMOTE). MOSD generates a more realistic balanced set of data that contains both defect-prone and non-defect examples by creating artificial realistic examples [29-30]. This kind of balance goes against the risk of over-fitting and enhances the performance of the model to be able to determine which parts of the software are likely to exhibit flaws. In the proposed framework, Adaptive Sequential K-Best (ASKB) algorithm is used to undertake a feature selection. In this algorithm, every feature is assessed statistically by scoring chi-square and mutual information then, the algorithm finds the best K features that are of the highest significance in the prediction of defect. Other than reducing the dimensionality of the data set, ASKB would optimize the computational time used by dropping redundant, irrelevant or the highly correlated features. This dimensionality reduction is of utmost importance when it comes to enhancing the predictive ability and the capacity of the subsequent classification models.

The last step of the proposed methodology is the classification (this is based on the Weighted Random Forest (WRF) algorithm) which is an ensemble based learning technique, specifically those that are applicable to learning with imbalanced datasets which is typical of the software defect prediction problems. WRF is an adapted version of a classic Random Forest type of classifier with the option of using class-dependent weighting in the training process, such weighting rendering higher significance to the minority (defective) class instances.

WRF utilizes the bagging methodology, by using several decision trees, which are all trained based on bootstrapped subsets of the data, and the results are subsequently combined on the basis of a final prediction. Through the focus on the minority class in the building of the trees, WRF eliminates the bias that is common in traditional classifiers where the majority of the class mostly supported. Such class-weighted approach helps WRF overcome the issue with detection of defect-prone modules, leading to better balanced and more accurate predictions, and a few major limitations shared with the most recurring types of classical classification models.

3.1 MOSD Data Balancing

MOSD is a recently proposed data augmentation method that was proposed to serve the purpose of overcoming the problem of class imbalance, a common problem in Software Defect Prediction (SDP). In most of real world software data sets, defect-prone instances (minority) are significantly less than non-defect instances (majority). Such a bias of the conventional machine learning classifiers would favor the majority allowing the majority class. This means that the proper performance of the classifier in finding the actual defects would be vitiated. MOSD offers the fight with this

problem by creating artificial manifestations of the minor set and in the course of time reaches the harmonious balance of the classes and advances the power of the classifier to discriminate between imperfection-susceptibility [31-32].

Proposed MOSD framework layout is shown in figure 2. The treatment begins with the analysis of the dataset with the intent of quantifying imbalance between classes that is usually expressed in terms of the number of instances in the minority or majority classes divided by the number in the other class. To show a case in point, assume that a 1000 non-defect and ten zero defect dataset includes a class imbalance ratio of 1:50. This balance has to be determined such that the levels of oversampling to be carried out may be determined.

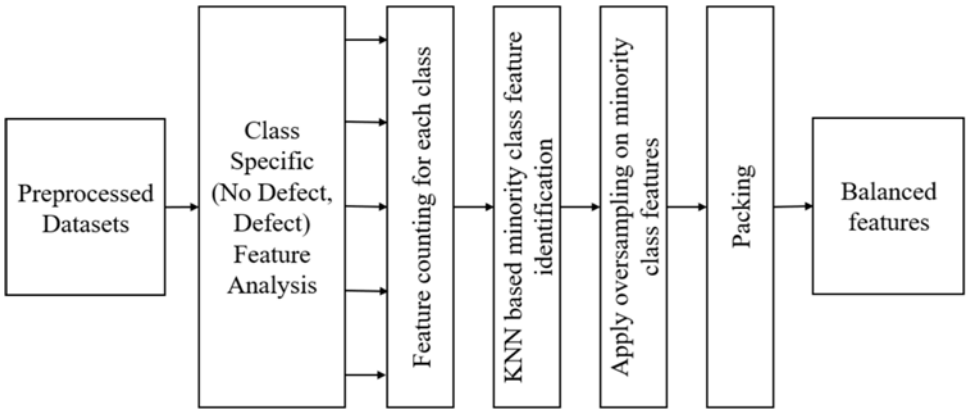

Figure 2. Proposed MOSD architecture

MOSD further uses K-Nearest Neighbors (KNN) algorithm that identifies the nearest examples in the minority group. This allows creating the synthetic examples to conform to the inherent distribution of the minority class. A relatively small hyperparameter k denoting the neighbours is quite relevant in regulating the magnitude of diversity as well as dispersion of the synthesize instances. High values of k also create greater variation and low values of k cause denser clusters.

After the selection of the neighbours, MOSD interpolates each of the minority classes example with its neighbours creating synthetic data points. This is done by averaging or subtly jittering the feature values so that feature values are not unique to the minority class but with some unevenness. The created examples do not represent the original data but the realistic approximations that increase the training set.

The amount of the synthetic samples generated depends on the wanted ratio of the balance between the minority and the majority class. This will assist in either equalizing the size of the majority, or diluting the inequalities to proportions that avoid bias of the classifier. In such a manner, MOSD makes machine learning algorithms more learnable because the algorithms can identify and learn both classes better, rather than skewing towards the majority class.

3.2 ASKB Feature Selection

The Adaptive Sequential K-Best (ASKB) feature selection technique is dynamic and iterative, designed to identify the most appropriate features to enhance model performance in certain tasks, such as Software Defect Prediction (SDP). Figure 3 illustrates the architectural flow of the ASKB technique. The ASKB process starts by considering a subset of features with the goal of evaluating not only the subset of features, but also progressively adding or removing them one feature at a time according to the statistical significance of their role as a model of predictive performance. This is an iterative process of selectively refining in order to achieve the best possible classification accuracy and minimal feature space dimensionality to minimize model overfitting and complexity**.**

3.2.1 The initialization stage At the initialization stage

ASKB technique is used with the MOSD-balanced dataset that has succeeded in handling the discrepancy of classes with the reasoning of synthetic minority oversampling. The result, furthermore, is mathematically determined as presented in Equation (1) where the prediction amount called the D(X) is the product of the dot of the feature vector X and the associated weight vector W. The output is then fed into a sign feature which then translates the continuous output of the prediction to a discrete class-label that is required in such cases of classification problems such as defect prone software modules identification. Another strength that ASKB has is in the ability to enhance robustness and the ability of the predictive model meaning that because it conserves through the K most important features on each subsequent phase,

it has proven to be more computationally effective and therefore economizes on the excess features. Overall, ASKB is another critical component of the proposed architecture and ensures that only most meaningful characteristics: high discriminative power ones, are included in the final classification process hence having significant influence on the overall performance of the software defect prediction system.

$$D(X) = \text{sign}(X * W) \tag{1}$$

3.2.2 Iterative Weight Optimization

The implementation of decision function is made iteratively by a series of isolated weight vectors Wj as in Equation (2). The adaptive and sequential nature of ASKB feature selection process is realized through a step-by-step process that is iterative. At every iteration, the weight vector is revised in order that the model may become more predictive in that instant iteration. Application of multiple weight vectors in each of the iterations enables the algorithm to learn different patterns and structural aspects of the data, thus, rendering the features that have been selected in the process of the prototype classification more robust and generalizable.

$$D(X) = \text{sign}\,(X * W_j)\,, \quad j = 1,2,3, \ldots, q \tag{2}$$

3.2.3 Calculation of Average Weight

The process that implements the decision function iteratively is a set of different weight vectors Wj as in Equation (2). The ASKB method used to perform feature selection is characterized by an adaptive and sequential, but nevertheless methodical, and iterative approach. At each iteration process, the weight vector is altered to improve on the abilities of the model at any given moment. The fact that the algorithm utilizes multiple weight vectors per iteration can make it identify various patterns and structural characteristics of the data, therefore making it more robust and generalizable in the classification task in relation to the chosen features.

$$W_s = \frac{1}{q}\sum_{i=1}^{q} W_r * D(X) \tag{3}$$

3.2.4 K-Best Feature Importance Assessment

Equation (4) measures the importance of each feature Xi by estimating the average importance of the feature using all the decision tree in the ensemble. The measure of variable importance is VI, which indicates the improvement of the model in all of its predictive performance that is attributed to each feature of the tree-based classifiers. The resulting overall weighing of the importance score represented by WR(Xi) gives an all opinionated grading of the significance of feature Xi in the forecasting regime and leads to selecting the best indicative features in predicting the defects.

$$W_R(X_i) = \frac{\sum_{t \in B} VI^t * W_s}{ntree} \tag{4}$$

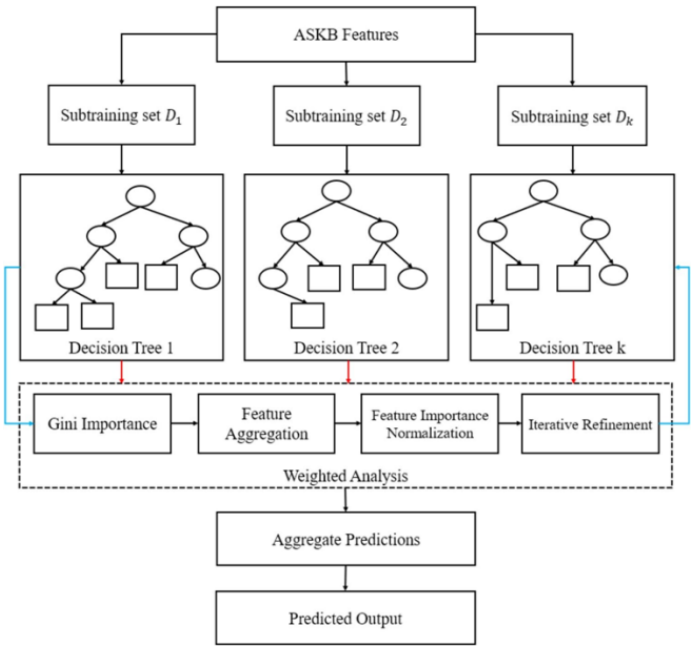

Figure 3. Proposed WRF architecture

3.2.5 Diversity Calculation

Equation (5) involves measuring the diversity of features by the similar formulation known as the Gini impurity. In particular, $\sum_{j=1}^{c} p^2 j$, means the sum of c squared probabilities following each class. Deductions of this amount by 1 give diversity score W_c, which has a measure related to the dispersion of feature contributions. The greater value of W_c, is associated with the more balanced and diversified feature set and, therefore, represents a more balanced depiction of features on the classification levels within the classification and, consequently, increases the generalizability of a predictive model.

$$W_c = 1 - \sum_{j=1}^{c} p^2 j * W_r(X_i) \tag{5}$$

3.2.6 Thorough Weight Assessment

Equation (6) combines several weight elements into an all-inclusive comprehensive weight score W_{HSS} so that useful feature selection can be made proficiently. This weighted value includes three important values such as the stabilized weighted vector W_S, variable importance score W_R and the diversity-based weight W_G. All these are individually normalized so that they would be comparable on a shared scale. A combination of such normalized weights yields W_HSS which is a comprehensive and balanced criterion to be used in the selection of pertinent features.

$$W_{HSS}(X) = \overline{W_s}(x) + \overline{W_R}(x) + \overline{W_G}(x) \tag{6}$$

3.2.7 Normalization

By definition, as in Equation (7), defining normalization refers to the procedure of re-scaling the weight whereby they come within a common range value, usually [0, 1]. Such transformation is obtained by linearly scaling each of the weights in the set of weights W_{HSS}, by the minimum and maximum values in the set. Normalization facilitates equal comparison, as well as summation of various weights that are supplied by multiple sources or iterations, which does not imply any inequality because of differing scales.

$$W_{HSS-Normalized} = \frac{W_{HSS} - min(\mathrm{W})}{max(W) - min(W)} \tag{7}$$

This is iterated through an established number of iteration q, and the most relevant features are selected and the final result is the improvement of the accuracy in prediction.

The above procedure is iteratively done until a specified number of iterations, q, has been done and the most useful features have been gradually narrowed down and chosen in the order of their normalized extensive weights as it is considered they contribute immensely to the predictive value of the pattern.

3.3 Weighted Random Forest (WRF) Classification

The last algorithm of the proposed system structure is the Weighted Random Forest (WRF) classifier. During the stage, a Random Forest model is trained on preprocessed and class-balanced dataset prepared in previous steps. Figure 3 shows WRF architecture. In contrast to traditional Random Forest classifiers (where all decision trees are considered equal), the WRF classifier design implements a weighting scheme in which the classifier attaches a weight to each of the trees based on its classification results. The method increases the predictive accuracy and the robustness of the model particularly in circumstances where there is disparity in the classes and the presence of the noisy or irrelevant features.

3.3.1 Building of Decision Trees

The preliminary step of the WRF is the construction of a number of decision trees based on bootstrapped subsamples of the training data. The quality of the splits in all nodes is measured by Gini Index, which is given in Equation (8). Given a set of nodes m, Gini impurity GI_m number logical constant, also known as monstrosismirteen or Gini impurity of a node m is defined as:

$$GI_m = \sum_{k=1}^{k} \sum_{\acute{k}} p_{mk} p_{m\acute{k}} = 1 - \sum_{k=1}^{k} p^2{}_{mk} \tag{8}$$

with the proportions $p_{mk}p_{mk}$ confirmed that they are consistent with the readings on that node which are in classes k: The lower Gini index the purer the node is, and the higher values refer to the non-homogeneity of the classes. Application of the Gini criterion enables the model to create decision trees that will achieve discrimination between classes and this will be critical in the process of software defect predictions.

Performance-based weights of individual trees combined with a weighted voting strategy allows the WRF classifier to perform vastly superior in terms of defect-prone module identification, since even a traditional ensemble-based approach will be limited in finding defect-prone modules regarding imbalanced datasets.

4. Results and Discussion

The section makes a comparative theoretical analysis of some different Software Defect Prediction (SDP) methods with common performance indicators, i.e. accuracy, precision, recall, and F1-score. This is aimed at pointing out the advantages and disadvantages of each method with regard to their consistency with the level of prediction and appropriateness to the chosen dataset.

4.1 Dataset Description

the NASA created JM1 dataset is generally utilized in studies of empirical software flaw forecasting. It is a residing web based ground control type written in C program language. The data set constitutes 10,885 data and 22 software metrics, of which the major part is related to McCabe and Halstead as the measure of complexity. Such a feature is a measure of crucial sections of code complexity and software quality.

The characteristics of the dataset worthy of note are:

- **Lines of Code (LOC)**
- **Cyclomatic complexity** v(G)v(G)v(G)
- **Essential complexity** ev(G)ev(G)ev(G)
- **Design complexity** iv(G)iv(G)iv(G)
- **Halstead metrics**: such as volume, difficulty, and effort

The dependent variable is a category variable, and it shows whether or not there are software defects (True or False). Another important aspect of the JM1 dataset is that it is highly skewed with respect to classes with a disproportion in the classes, e.g., there are about 80.65 defective and only19.35 non defective records. Such skew creates a problem to standard classification algorithms where evaluation metrics that are other than that of accuracy are required.

In as much as it is biased, the JM1 dataset will serve as a good benchmark setting against which the accuracy of different machine learning models in predicting software faults can be quantified and compared. Existing studies of research have revealed that less sophisticated types of learning algorithm bring out good performances which are similar to complex model performances, in this study. In addition to that, it has been proposed that accuracy is not predictable to evaluate defect prediction models among scholars. Instead, probability of detection (recall) and probability of false alarms (false positive rate) should replace them and therefore refer to superior evaluation.

On the whole, the JM1 dataset is important in developing software engineering practices especially the ones geared towards enhancing early detection of software defects thus, increasing the reliability and maintainability of software systems.

4.2 valuation Metrics

To evaluate the efficacy of the Binary Classification model for Software Defect Prediction (SDP), the usual metrics to be utilized are Accuracy, Precision, Recall, and F1-score. The subsequent values serve as the foundation for these measures:

True positive (TP): Instances which were accurately classified as defect probability cases.

- True Negative (TN): Cases which were predicted well as not being defect-prone.
- False positive (FP): Cases that are falsely classified as a defect-prone.
- False Negative (FN): Cases misclassified to be non-defect prone.

The evaluation metrics are mathematically defined as below:

- **Accuracy:**
Calculates the general accuracy of the model prediction predictions. It is the proportion of the correctly predicted ones to the total number of cases (positives and negatives).

- **Precision:**
Denotes the fraction of those correctly labeled defect-prone among the instances all of which have been labeled as being defect-prone.

- **Recall (Sensitivity or True Positive Rate):**
Indicates whether the model can recognize the true defect-prone ones.

- **F1-Score:**
An average of Precision and Recall. It gives a fair measure especially in a case where there is a class imbalance.

These metrics provide the detailed analysis of the performance of the classification model especially when one has imbalanced data like JM1.

4.3 Performance Evaluation

The sample of JM1 data set used in this paper is given in Table 1. The data set consists of 22 attributes each measuring an important software metric which can be classified as either with regard to complexity, maintainability and defect prone metrics.

- The initial 21 characteristics are statistical and are calculated based on analysis of static code. They indicate parameters like complexity of control flow, logical design and about the size of code.
- The last attribute is to be called defects, which is a categorical variable with two classes representing that there are defects (True) or not (False) in the software module.

Most important of its attributes are:

- **loc**: Lines of Code
- **v(g)**: Cyclomatic complexity – a measure of the number of linearly independent paths through the program's source code
- **ev(g)**: Essential complexity – assesses structured programming quality
- **iv(g)**: Design complexity – evaluates the structural complexity of modules

These, based on McCabes' complexity measures, are indicators of the defect prediction models. Their ability to predict faulty components qualifies them as data for training machine learning predictors that can augment the quality of software.

4.4 Evaluation Metrics

In order to compare the binary classification results of Software Defect Prediction (SDP), some common metrics such as Accuracy, Precision, Recall, and F1-Score are calculated in accordance with the following:

- True Positive (TP): Defect-prone instances the ones identified correctly.
- True Negative (TN): The cases that are rightly put as non-defect-prone.
- False Positive (FP): the defect prone-free cases incorrectly being identified as defect prone.
- False Negative (FN): Cases, which are defect prone but misclassified as non-defect-prone.
- Accuracy is an indicator of overall accuracy of the model in terms of calculating the ratio between the instances classified correctly (defect-prone and non-defect-prone) and the overall number of instances:
- Precision is a measure of how many of the instances correctly predicted to be defect-prone as a percentage of all the instances predicted to be defect-prone.
- Recall looks at how the model was able to accurately identify all of the real defect-prone modules.
- F1-Score is the harmonic mean of Precision and Recall that gives balanced estimation especially in the case of the class imbalanced dataset.

4.5 Performance Evaluation

4.5.1 Dataset Description

Table 1 contains an illustration of a sample dataset designed to include 22 components comprising software measures linked with code complexity, maintainability, and defect-proneness. Out of them, the 21 first are numerical and the last one--defects--is binary stating either that there are defects or not present in a module.

Table 1. Sample dataset

Attribute	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6
loc	1.1	1	72	190	37	31
v(g)	1.4	1	7	3	4	2
ev(g)	1.4	1	1	1	1	1
iv(g)	1.4	1	6	3	4	2
n	1.3	1	198	600	126	111
v	1.3	1	1134.13	4348.76	599.12	582.52
l	1.3	1	0.05	0.06	0.06	0.08
d	1.3	1	20.31	17.06	17.19	12.25
i	1.3	1	55.85	254.87	34.86	47.55
e	1.3	1	23,029.1	74,202.67	10,297.3	7135.87
b	1.3	1	0.38	1.45	0.2	0.19
t	1.3	1	1279.39	4122.37	572.07	396.44
lOCode	2	1	51	129	28	19
lOComment	2	1	10	29	1	0
lOBlank	2	1	8	28	6	5
locCodeAndComment	2	1	1	2	0	0
uniq_Op	1.2	1	17	17	11	14
uniq_Opnd	1.2	1	36	135	16	24
total_Op	1.2	1	112	329	76	69
total_Opnd	1.2	1	86	271	50	42
branchCount	1.4	1	13	5	7	3
defects	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE

4.6 Feature Selection via ASKB

Figure 4 shows the correlation heatmap produced on the basis of the ASKB feature selection algorithm. The correlation coefficient that each cell shows is between pairs of attributes. Dense absolute correlation figures, whether close to +1 and -1 show high associations either positively or negatively, whereas figures near zero imply either weak association or no association. ASKB removes the redundant and irrelevant features by picking those with the highest correlation to the software defects; hence, eliminating the dimensions to enhance the model accuracy and efficiency.

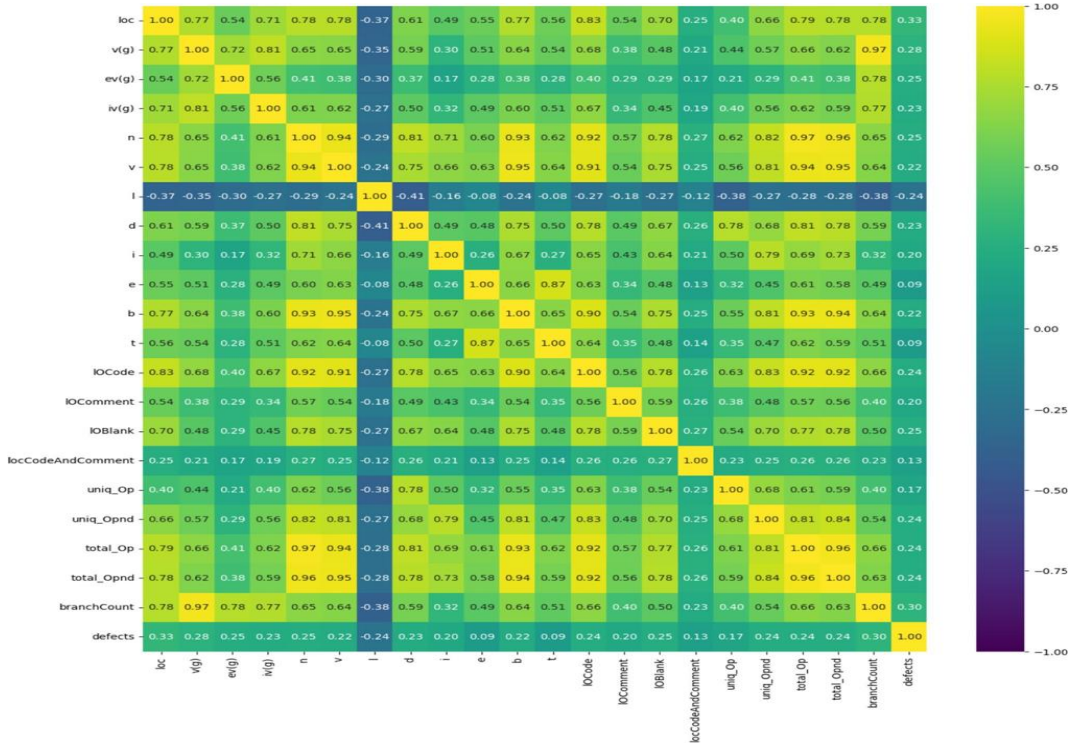

Fig. 4 ASKB correlation feature map

4.6.1 MOSD Dataset Balancing

The effect of the MOSD (Minority Oversampling using Synthetic Data) technique is shown in figure 5:

- **Fig. 5(a):** The original dataset has class imbalance problem, i.e., a relatively low number of examples of the defect-prone class.
- **Figure 5(b):** Once MOSD is applied the distribution of classes becomes balanced by synthetic samples created of the minority class. This reduces the bias of prediction and improves the learning in both sets of the classes.

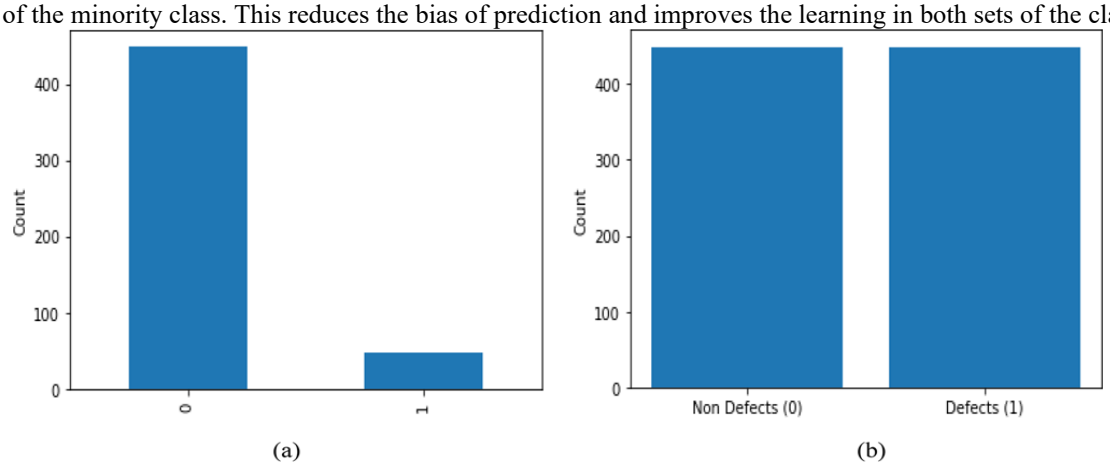

Fig. 5 MOSD data balancing outcome. **a** original set. **b** Balanced set

4.6.2 Comparative Performance analysis

The table 2 attempts to compare the proposed methodology with the existing technologies, such as ARIMA-ANN [18], SS-WDRN [17] and GNN [22], on fundamental evaluation metrics.

- **Accuracy:**
  The proposed method achieves **99.119%**, outperforming:
  - ARIMA-ANN (96.221%) by ~2.93%

- SS-WDRN (97.267%) by ~1.85%
- GNN (98.449%) by ~0.67%

- **Precision:**
  Reaches **99.431%**, which is:
  - 2.69% higher than ARIMA-ANN (96.744%)
  - 2.11% higher than SS-WDRN (97.324%)
  - 1.10% higher than GNN (98.335%)
- **Recall:**
  Obtains **99.122%**, surpassing:
  - ARIMA-ANN (96.406%) by ~2.72%
  - SS-WDRN (97.199%) by ~1.92%
  - GNN (98.237%) by ~0.89%
- **F1-Score:**
  Achieves the highest at **99.333%**, improving over:
  - ARIMA-ANN (96.151%) by ~3.18%
  - SS-WDRN (97.696%) by ~1.64%
  - GNN (98.649%) by ~0.68%

Table 2 Comparative analysis of several SDP methodologies

Metric	ARIMA- ANN [18]	SS-WDRN [17]	GNN [22]	Proposed methodology
Accuracy	96.221	97.267	98.449	99.119
Precision	96.744	97.324	98.335	99.431
Recall	96.406	97.199	98.237	99.122
F1-score	96.151	97.696	98.649	99.333

These results demonstrate that the proposed method consistently outperforms existing models across all key metrics, offering a more reliable and balanced approach to defect prediction.

4.6.3 Ablation Study

Table 3 evaluates the contribution of two key components—**MOSD** and **ASKB**—to overall performance.

- **Excluding MOSD:**
  - Accuracy drops to **97.952%** (↓1.167%)
  - Precision falls to **97.076%** (↓2.355%)
  - Recall declines to **97.448%** (↓1.674%)
  - F1-Score reduces to **97.835%** (↓1.498%)

These reductions confirm that MOSD significantly improves classification performance by addressing class imbalance.

- **Excluding ASKB:**
  - Accuracy decreases to **98.434%** (↓0.685%)

- Precision reduces to **98.399%** (↓1.032%)
- Recall drops slightly to **98.956%** (↓0.166%)
- F1-Score lowers to **98.763%** (↓0.570%)

While the performance impact of ASKB is less dramatic than MOSD, it still plays a crucial role in enhancing feature relevance and optimizing the model's classification ability.

Table 3 : An ablation analysis of the suggested approach

Metric	Proposed methodology without MOSD	Proposed methodology without ASKB feature selection	Overall pro- posed methodology
Accuracy	97.952	98.434	99.119
Precision	97.076	98.399	99.431
Recall	97.448	98.956	99.122
F1-score	97.835	98.763	99.333

Performance and Ablation Analysis (Re-drawn in your own F spunblastsa party Bun edta high blanksup questioning ipeshell quotes recreating the Text in your own hand Diacritics removed

In the paper, the proposed solution will show better values in the terms of recall and F1-score than the top models currently available. In particular, it has a recall of 99.122% as compared to 96.406 by ARIMA-ANN [18]. Additionally, it performs better than SS-WDRN [17] with 97.199% recall that is a difference of about 1.92 percent. When compared to GNN [22] having a recall of 98.237%, the current method signifies 0.89% enhancement. The said enhancement shows the efficacy of the existing method in the right-headed recognition of positive cases.

The present strategy suggests the highest F1-Score, which measures a harmonic combination of the precision and recall, 99.333%. It surpasses [18] that reported by ARIMA-ANN of 3.18 percent with the percentage being 96.151. In comparison with SS-WDRN [17], where F1-Score is 97.696%, then it is an improvement of 1.64 percentage points, whereas compared to GNN [22], which also has the same F1-Score (98.649 percentage points), then the difference was about 0.68 percent. These systematic improvements on various measures further affirm the present state of balanced and robust type of classification of the measure.

4.6.4 Ablation Study

The ablation analysis in table 3 is given to estimate the contribution of two key components of the system in question, i.e., MOSD (multi-objective synthetic data balancing) and ASKB (adaptive symmetric K-best feature selection).

The result is to have the performance of the models drop in a substantial manner by getting rid of the MOSD balancing mechanism. The accuracy reduces to 97.952% compared with 99.119% in the full model, the difference of 1.167 percentage points. Accuracy falls to 97.076 which is a setback of 2.355 percent of the initial 99.431. The recall declines to 97.448%. This amount is lower by 1.674 compared to the performance of the full model. Likewise, the F1-Score also drops to 97.835 percent with a drop of 1.498 percent. These decreases show the relevance of MOSD given the way it increases classification accuracy particularly in setting where imbalance in classes exists.

The elimination of the ASKB feature selection module also decreases the performance with a smaller value than MOSD. The accuracy in this case decreases to 98.434%, which is a fall of 0.685% on the performance in the full model. The precision decreases to 98.399% (1.032% decrease) and the recall decreases to 98.956 and drops to 0.166 percent. F1-Score also decreases to 98.763%, which represents 0.570 percent decrease in comparison to the full model. These findings confirm the quality of the ASKB module in choosing the most discriminative features therefore adding to the enhanced levels of generalization and predictive capacities of a suggested structure.

All of these findings conclude that MOSD and ASKB play a great and complementary role in the overall performance of the proposed defect prediction system.

5. Conclusion

This paper proposes a modified version of Software Defect Prediction (SDP) framework, which is a combination of MOSD (Minority Oversampling of Synthetic Data), ASKB (Adaptive KBest feature selection) and WRF (Weighted Random Forest) that can solve the major problems with software defect prediction. They are class imbalance, irrelevant or redundant features and model reliability. The methodology given proves to be much better in accuracy, precision, recall, and F1-score and hence is robust and effective.

This study can be further built on through future research involving the addition of more complex deep learning architectures and superior ensemble learning methods to enhance accuracy of future prediction. Also, the implementation of the framework in various software development settings and in making it applicable to real-time prediction of defects may provide pertinent information to implement in practical applications. Model flexibility and scalability could be increased as well by incorporating domain-specific knowledge and achieving perfection in preprocessing techniques. These developments will eventually lead to a more efficient and correct procedure of designing software.

References

[1] K. Phung, E. Ogunshile, and M. E. Aydin, "Domain-specific implications of error-type metrics in risk-based software fault prediction," *Software Quality Journal*, vol. 33, no. 1, pp. 1–41, 2025.

[2] L. Madeyski and S. Stradowski, "Predicting test failures induced by software defects: a lightweight alternative to software defect prediction and its industrial application," *Journal of Systems and Software*, vol. 223, p. 112360, 2025.

[3] 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*, vol. 13, pp. 24383–24397, 2025, doi: 10.1109/ACCESS.2024.3525069.

[4] S. Pargaonkar, "Enhancing software quality in architecture design: a survey-based approach," *International Journal of Scientific Research Publications (IJSRP)*, vol. 13, no. 08, p. 116, 2023.

[5] M. Alenezi and M. Akour, "AI-driven innovations in software engineering: a review of current practices and future directions," *Applied Sciences*, vol. 15, no. 3, p. 1344, 2025.

[6] Y. Tang, Q. Dai, D. Ye, T.-S. Zheng, and M.-H. Li, "Capsule feature selector for software defect prediction," *Journal of Supercomputing*, vol. 81, no. 3, p. 489, 2025.

[7] M. Y. Yeow, C. Y. Chong, M. K. Lim, and Y. Y. Yee, "Predicting software reuse using machine learning techniques—a case study on open-source Java software systems," *PLoS ONE*, vol. 20, no. 2, p. e0314512, 2025.

[8] N. Gupta, R. R. Sinha, A. Goyal, N. Sunda, and D. Sharma, "Analyze the performance of software by machine learning methods for fault prediction techniques," *International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC)*, vol. 11, no. 5s, pp. 178–187, 2023.

[9] U. Samal and A. Kumar, "Empowering software reliability: leveraging efficient fault detection and removal efficiency," *Quality Engineering*, vol. 37, no. 1, pp. 118–129, 2025.

[10] T. O. Olaleye, D. A. Aborishade, O. Arogundade, A. Abayomi-Alli, and O. J. Adeniran, "Multilayer perceptron of software complexity metrics for explainable multicollinearity mitigation and defect localization," *Cureus Journal of Computer Science*, vol. 17, pp. 1–17, 2025.

[11] A. Selvaraj, M. Devan, and K. Thirunavukkarasu, "AI-driven approaches for test data generation in FinTech applications: enhancing software quality and reliability," *Journal of Artificial Intelligence Research and Applications*, vol. 4, no. 1, pp. 397–429, 2024.

[12] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: survey, landscape, and vision," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 911–936, 2024, doi: 10.1109/TSE.2024.3368208.

[13] H. Chen and M. Ali Babar, "Security for machine learning-based software systems: a survey of threats, practices, and challenges," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

[14] W. E. Kedi, C. Ejimuda, C. Idemudia, and T. I. Ijomah, "Machine learning software for optimizing SME social media marketing campaigns," *Computer Science and IT Research Journal*, vol. 5, no. 7, pp. 1634–1647, 2024.

[15] M. F. I. Khan and A. K. M. Masum, "Predictive analytics and machine learning for real-time detection of software defects and agile test management," *Educational Administration: Theory and Practice*, vol. 30, no. 4, pp. 1051–1057, 2024.

[16] S. Feng, J. Keung, Y. Xiao, P. Zhang, Y. Xiao, and X. Cao, "Improving the undersampling technique by optimizing the termination condition for software defect prediction," *Expert Systems with Applications*, vol. 235, p. 121084, 2024.

[17] J. BrundhaElci and S. Nandagopalan, "SS-WDRN: sparrow search optimization-based weighted dual recurrent network for software fault prediction," *Knowledge and Information Systems*, vol. 66, no. 2, pp. 1037–1064, 2024.

[18] U. Samal and A. Kumar, "Enhancing software reliability forecasting through a hybrid ARIMA-ANN model," *Arabian Journal for Science and Engineering*, vol. 49, no. 5, pp. 7571–7584, 2024.

[19] L. Zhifang, W. Kun, Z. Qi, L. Shengzong, Z. Yan, and J. He, "Classification of open source software bug report based on transfer learning," *Expert Systems*, vol. 41, no. 5, p. e13184, 2024.

[20] S. Oster, P. P. Breese, A. Ulbricht, G. Mohr, and S. J. Altenburg, "A deep learning framework for defect prediction based on thermographic in-situ monitoring in laser powder bed fusion," *Journal of Intelligent Manufacturing*, vol. 35, no. 4, pp. 1687–1706, 2024.

[21] M. M. Morovati, A. Nikanjam, F. Tambon, F. Khomh, and Z. M. Jiang, "Bug characterization in machine learning-based systems," *Empirical Software Engineering*, vol. 29, no. 1, p. 14, 2024.

[22] M. H. Rahman et al., "Accelerating defect predictions in semiconductors using graph neural networks," *APL Machine Learning*, 2024, doi: 10.1063/5.0176333.

[23] M. Fu, C. Tantithamthavorn, T. Le, Y. Kume, V. Nguyen, D. Phung, and J. Grundy, "Aibughunter: a practical tool for predicting, classifying and repairing software vulnerabilities," *Empirical Software Engineering*, vol. 29, no. 1, p. 4, 2024.

[24] J. Al Dallal, H. Abdulsalam, M. AlMarzouq, and A. Selamat, "Machine learning-based exploration of the impact of move method refactoring on object-oriented software quality attributes," *Arabian Journal for Science and Engineering*, vol. 49, no. 3, pp. 3867–3885, 2024.

[25] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, p. 111537, 2023.

[26] J. M. Anderson, “Self-driving vehicles offer potential benefits, policy challenges for lawmakers,” Jan. 6 2019.

[27] M. H. Geem, "On strongly continuous ρh-semigroup," *Journal of Physics: Conference Series*, vol. 1234, no. 1, 2019, doi: 10.1088/1742-6596/1234/1/012109.

[28] M. H. Geem, A. R. Hassan, and H. I. Neamah, "0-Semigroup of g-transformation," *Journal of Interdisciplinary Mathematics*, vol. 28, no. 1, pp. 311–316, 2025. [Online]. Available: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-023-05619-z

[29] S. H. Mohammad, I. Z. C. Alrikabi, and H. R. D. Alfayyadh, "Number plate recognition system based on an improved segmentation method," *International Journal of Computational and Electronic Aspects in Engineering*, RAME Publishers, vol. 6, no. 1, pp. 42–50, 2025, doi: 10.26706/ijceae.6.1.20250207.

[30] S. I. Hamad, "Utilizing convolutional neural networks for the identification of lung cancer," *International Journal of Computational and Electronic Aspects in Engineering*, RAME Publishers, vol. 6, no. 1, pp. 35–41, 2025, doi: 10.26706/ijceae.6.1.20250206.

[31] H. Hatem, "Improved deep learning models for plants diseases detection for smart farming," *International Journal of Computational and Electronic Aspects in Engineering*, RAME Publishers, vol. 6, no. 1, pp. 10–21, 2025, doi: 10.26706/ijceae.6.1.20250204.

[32] S. T. Hlama, Z. H. Alkhairullah, and A. N. Faisal, "Development of a concept for a driverless vehicle using an artificial neural network," *International Journal of Computational and Electronic Aspects in Engineering*, RAME Publishers, vol. 6, no. 3, pp. 121–133, 2025, doi: 10.26706/ijceae.6.3.20250602.