Minimax Theory and its Applications

Volume 11 (2026), No. 3s, 868 to 887

An Adaptive Cluster-Guided Ensemble Learning Framework for Robust software defect prediction and Overfitting Mitigation

Srinivasa Rao Katragadda[1], Dr.Sirisha Potluri[2]

[1]Research scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Bowrampet, Hyderabad-500043, Telangana, India.Email: ksrresearch1@gmail.com
[2]Associate Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Bowrampet, Hyderabad-500043, Telangana, India. Email: Sirisha@klh.edu.in

Abstract
Software defect prediction (SDP) improves testing efficiency, maintenance planning, and quality assurance by identifying fault-prone modules before deployment. However, existing approaches often struggle with heterogeneous software modules, redundant correlated metrics, class-imbalanced defect data, and overfitting caused by excessive adaptation to training observations, which limit generalization to unseen modules. To address these challenges, this paper proposes an Adaptive Cluster-Guided Ensemble Learning Framework (ACGELFW) built around the Mitra-v2 tabular foundation model. The framework preprocesses software-metric data and applies correlation-based feature optimization to obtain a compact, non-redundant representation. Adaptive K-Means clustering, guided by silhouette analysis, groups structurally similar software modules, and the resulting cluster identity is supplied as contextual information to a shared Mitra-v2 model for defect probability estimation. Overfitting is controlled through L2 regularization, validation-based early stopping, and stratified partitioning across training, validation, and test sets. Cluster-level predictions are combined using validation-derived weights to produce an aggregated defect probability, thresholded for final binary classification. Experiments on a combined dataset of 14,625 software modules from JM1, KC1, CM1, KC2, and PC1 show that silhouette analysis identifies four homogeneous clusters, while feature optimization reduces the metric set from 20 to 15 attributes. The proposed framework achieves a testing accuracy of 97.64%, precision of 97.51%, recall of 97.82%, F1-score of 97.66%, MCC of 95.31%, ROC-AUC of 98.18%, and PR-AUC of 97.94%, outperforming Random Forest, PCA-LDA with IMJAYA-ELM, and Bi-LSTM baselines. These findings confirm that combining adaptive clustering with a single shared foundation model yields robust, generalizable defect predictions without requiring separate classifiers for each module group.

Keywords: Prediction, Ensemble Learning, Adaptive Clustering, Overfitting Mitigation, Tabular Foundation Model

1. Introduction

The SDP research area is significant in software engineering because predicting software defects in early stages can help in improving the effectiveness of software testing, maintenance and quality assurance[1]. Many metrics have been developed at structural and programming levels for software systems such as lines of code, complexity, coupling, operators, operands, etc., that can be used to describe a software module[2]. Machine-learning techniques can capitalize on these metrics and detect patterns that are related to defective and non-defective modules[3]. But software defect data sets are usually heterogeneous and, modules that come from different projects, may have very different distributions of metrics and characteristics of defects. Therefore, learning a single global decision boundary might not be enough to capture the different structural patterns in the data. Furthermore, redundant features may make the model more complex, and cause extra learning variations, impacting on prediction accuracy[4].

The opportunities to learn complex relationships from structured software metrics have been made possible by recent advances in tabular AI. However, the application of defect prediction is still difficult due to the following reasons: First, the characteristics of each module are different; second, there are a lot of redundant attributes; third, the defect data is imbalanced; and fourth, some defects are not observed during the training process. In addition, too many adaptations to the training observations can cause overfitting, in which the model can perform better on the observations it was trained on, but not as well to unseen software modules. The challenges signify the necessity for a prediction framework that can be capable of pinpointing groups of modules that have

comparable characteristics, and preserve the correct controls while training the model[5]. The use of structural similarity in the learning process might be useful to give further context information to help separate defect prone modules.

This paper presents an Adaptive Cluster-Guided Ensemble Learning Framework for Robust SDP and Overfitting Mitigation (ACGELFW) based on Mitra-v2 as the core Artificial Intelligence model to solve these problems. The proposed framework initially processes the data and optimises the features to get a reliable and compact software-metric representation[6]. The software modules are then clustered using Adaptive K-Means method and groups of software modules are identified that have similar characteristics. The cluster information is fed to context as contextual information during Mitra-v2 based defect prediction. Regularization, validation monitoring and early stopping are added to the training process to minimize excessive fitting. Finally, the outputs of the predictions at the cluster level are aggregated based on the weights determined from the validation results, to obtain final defect probability and defect classification. The framework is thus combined into a single SDP pipeline with feature optimization, adaptive clustering, contextual tabular learning, overfitting control and ensemble aggregation.

1.2Problem Statement

The SDP approaches often face various challenges such as the presence of heterogeneous software modules, multiple measures, differing distributions and limited observation of software defects[7]. A traditional global prediction model assumes the representation of all software modules in the same manner and hence may not be able to account for the differences between structurally different groups of modules[8]. If there is high correlation and redundancy among the features, it can further complicate the model and give it unnecessary information when it's learning. Moreover, the model can overfit if it over-adapts to the training modules, thus limiting its ability to generalize well to unseen modules. This has motivated the development of an adaptive defect prediction framework that can take advantage of the similarity between software modules and yet not be too complex[9]. The proposed study tackles this problem by combining feature optimization, K-means based module clustering, cluster guided Mitra-v2 prediction, overfitting control and cluster level ensemble aggregation to a single prediction framework[10].

1. 3 Research Motivation

The motivation for this study is to advance the defect prediction problem for heterogeneous software modules, by devising an effective structured-data learning approach. Clustering may point out similar defect patterns among the software modules in terms of metric characteristics to give useful context information for prediction. Furthermore, a systematic feature optimization and an effective overfitting control are also vital in the model development for a stability and a better generalization on new software modules and features.

1. 4 Research Significance

The importance of this research work is that in one single SDP system feature optimization, adaptive clustering, prediction using Mitra-v2, overfitting control, and cluster-level ensemble aggregation have been integrated. The proposed method offers a systematic way of integrating software-module similarity with a limit on model complexity when predicting defects. The combination can give a structured framework to analyze the heterogeneous software metrics and to build a strong system of SDP for software quality assurance.

1.5 Key Contributions

- Proposed an adaptive cluster-guided approach to robust SDP based on heterogeneous software metrics.
- Designed a defect prediction mechanism based on the Mitra-v2 approach that integrates cluster derived contextual information.
- Proposed an adaptive K-Means clustering approach to determine structurally similar software modules.
- Use of integrated regularization, early stopping and validation-based monitoring to prevent model overfitting.
- Designed an ensemble aggregating mechanism at the cluster level to combine contextual prediction results.
- Implemented a feature optimization pipeline to identify and remove redundant and highly correlated software metrics.

1.6 Organization of the Paper

The rest of this paper is organised as follows. In Section 2, a thorough survey of existing SDP techniques and related machine learning (ML) techniques is provided. The proposed methodology is explained in detail in section 3, which includes a description of the data pre-processing, feature optimization, clustering, prediction using Mitra-v2, overfitting control and ensemble aggregation. The results of the experimental evaluation are presented in Section 4. Section 5 summarizes the paper, and provides suggestions for future research.

2. Related Work

Althiban et al. [11] explored defect prediction in parallel programs using hybrid MPI and OpenMP, as deadlocks and race conditions are among the complex defects that are difficult to detect in HPC systems. The authors suggested to use a ML approach in predicting defects in C++ systems with a hybrid MPI/OpenMP programming model, specifically focusing on a balanced dataset for predicting with better reliability. The experimental study tested several ML classifiers with Term Frequency (TF) as the feature representation and RFE as the feature selection method. Support Vector Machine (SVM) classifier using TF and RFE showed highest accuracy and prediction performance amongst the evaluated approaches as compared to other classifiers considered. The study pointed out balancing the dataset and feature-selection techniques can contribute to reliable defect prediction. The approach was introduced as a way to detect defects earlier in the development process, to decrease development time and to enhance reliability of HPC software systems.

Alqarni and Aljamaan[12] studied SDP when the training data sets are imbalanced, where traditional machine-learning algorithms tend to over-represent the majority non-defective class. To tackle the problem of minority defective modules, the authors suggested that Generative Adversarial Network (GAN) can generate data and then combine with ensemble learning. The synthetic minority class samples and the balancing of the defect datasets were generated using the GAN-based methods, and the classification model using which the classification of the defective and non-defective modules was performed was AdaBoost. The proposed approach was evaluated by 10 software defect datasets of different imbalance ratios. Statistical comparisons of performance differences were made with the Wilcoxon effect size difference test and Scott-Knott effect size difference test. The experimental results show that GAN-based techniques need to be adjusted properly to be successfully applied to imbalanced SDP. The GAN-based methods outperform the traditional sampling methods for dealing with class imbalance. Moreover, the study concluded that there was no improvement in the use of undersampling with GAN-based methods, underscoring the need to choose suitable strategies for handling the imbalance.

Hernández-Molinos et al. [13] were interested in using Bayesian methods to predict programs prone to defects; they discussed the underutilisation of Bayesian Networks in previous studies of SDP. The authors compared the performance of three algorithms for the construction of a Bayesian Network: K2, Hill Climbing and Tree-Augmented Naive Bayes (TAN). Three public PROMISE datasets were used in the experimental study, which were derived from the metric of McCabe and Halstead software complexity. Cross-validation of the Bayesian methods with various performance measures was performed and they were compared with the frequently-used Decision Tree and Random Forest methods. The obtained results proved that the Bayesian Network models achieved classification performance similar to Decision Tree and Random Forest models. Further, the Bayesian methods showed reduced variations in the training and testing partitions. The authors pointed out that the decreased variance can help increase robustness in SDP as the model's accuracy was not affected by the choice of training and test data. The study shows how the Bayesian Networks can be used as an alternative method to reliable defect classification.

Ali et al. [14] presented an intelligent ensemble-based SDP model that can prioritize software modules for testing and minimize effort. The prediction process was designed in two steps, all classifiers were trained and optimized iteratively to enhance the prediction accuracy (Random Forest, Support Vector Machine, Naïve Bayes, Artificial Neural Network classifiers). The results of these classifiers were combined in the second stage by creating an ensemble of classifiers based on voting method to predict the final defects. To assess the proposed system, seven datasets of historical defects were supplied by NASA MDP repository, such as CM1, JM1, MC2, MW1, PC1, PC3, and PC4. Experimental results proved the high accuracy of the predictions made by the proposed approach for the different datasets tested and have also been compared with twenty different SDP approaches, such as individual classifiers or ensemble methods and have proved to yield better results. The results show that SDP using voting-based ensemble learning of multiple optimized classifiers can increase the metrics.

Petrovic et al. [15]conducted a study on the use of artificial intelligence (AI) and natural language processing (NLP) alongside the metaheuristic optimization technique for automated software defect detection, which is currently becoming more difficult with manual source code inspection. The authors suggested two-layer design in which they used a Convolutional Neural Network (CNN) to process the high-dimensional feature space, and classifiers including AdaBoost and XGBoost were applied to the task of defect identification. To enhance the performance of classifiers, the metaheuristic optimization techniques were added, and the performance of the representations obtained from the NLP was also investigated with the term frequency–inverse document frequency (TF-IDF) encoding. Five experiments were conducted, all of which were publicly available, to test the framework. CNN's model accuracy was 0.768799, AdaBoost model accuracy was 0.772166, and XGBoost model accuracy was 0.771044. The accuracies of XGBoost and AdaBoost improved with NLP-based processing. The results indicate the viability of using a representation method based on NLP and source-code analysis along with optimized ML methods for automated SD detection.

Javed et al. [16] explored the cross-project defect prediction (CPDP), dealing with problems resulting from distribution gap between source and target project distributions, class imbalance, noisy instances and lack of semantic information. The authors introduced a hybrid SMOTE-ENN technique for class rebalancing and noise reduction, and correlation-based feature selection (CFS) and best-first search (BFS) for selecting relevant features, named as SCAG-LSTM. The model also incorporates bidirectional gated recurrent units (Bi-GRU), bidirectional long short-term memory (Bi-LSTM), and an attention mechanism to model contextual dependencies and highlight significant features. F1-measure and AUC were used for comparison with a number of existing CPDP approaches to the proposed model. On AEEEM and SCAG-LSTM outperformed baseline results by up to 33.03%, 16.32% in F1-measure and AUC respectively, while on PROMISE they achieved up to 42.60%, 34.90% in F1-measure and AUC respectively. The study illustrates the feasibility of the domain adaptation and optimized recurrent learning for CPDP.

Arasteh et al. [17] studied the SDP to minimize the effect of an excess of training features on a classifier. The authors suggested Binary Chaos-based Olympiad Optimization Algorithm (BCOOA) for feature selection problem, which is an NP-hard problem, and came up with the most influential software metrics for defect classification. Features, selected in the above way, were then given to ML classifiers to identify the faulty software modules. Five real world software defect datasets and five standard software defect dataset were used for training and testing the proposed approach. In the 21 features that were available, six were found to be highly influential: basic complexity, number of operators, number of operands, number of lines of code and comments, and number of operands. The proposed model with a feature selection and machine-learning technique gave an high quality metrics. The results show that feature selection with the help of metaheuristics can eliminate irrelevant features and enhance software defect classification accuracy.

Haldar and Capretz[18] studied how project effort and static source-code metrics affect the prediction of software defects in an interpretable way, focusing on enhancing the explainability of the result of the prediction. The authors have applied SVM, Random Forest, k-NN and Artificial Neural Network techniques for developing the defect prediction models. To assess the reliability of the models across the different sources of data, the models were tested using a published data set within the project, as well as cross-project. To explore the impact of individual features on model predictions, the techniques of SHapley Additive exPlanations (SHAP) and Local Interpretable Model-Agnostic Explanations (LIME) were used. The findings showed that the results achieved were accurate in both independent and cross-project datasets, and that the static source-code metrics (such as software size and software complexity) have a significant role in the defect prediction process. The study also showed that explainability techniques could be used to gain knowledge about the model's decisions as well as make the SDP system more transparent.

Liao et al. [19] introducedthe method for using ML to quantum error mitigation (ML-QEM), which aimed to reduce the computational burden of error reduction in near-term quantum computers. The authors tested several machine-learning algorithms, such as linear regression, Random Forest, multilayer perceptron and graph neural network models, and trained them to learn error-mitigation strategies for a variety of quantum circuits and noise conditions on the device. Experiments were performed on the latest quantum computers with up to 100 qubits, with numerical evaluations under interpolation and extrapolation settings. For comparison, the digital zero-noise extrapolation method was used as a reference mitigation method. According to the study, the use of ML-QEM could significantly lower the runtime cost of quantum error mitigation without compromising the quantum error mitigation accuracy. The authors also suggested implementing the same mitigation strategies traditionally with ML with lower computational costs, which would represent a scalable mitigation approach. In summary, the results show that classical machine-learning methods can significantly cut down the overhead of mitigation and better enable practical quantum-computing applications.

Khleel and Nehéz [20] studied SDP and focused on the class-imbalance problem that might lead to the overfitting of a majority non-defective class in conventional classifiers. To solve the minority-class recognition problem, the authors suggested the CNN-based prediction model and GRU-based prediction model with training data balancing based on SMOTE Tomek sampling. Benchmark software defect databases from the PROMISE repository were employed to test the proposed approach. The metrics were used to evaluate the model performance. Experimental results revealed that balanced datasets achieved better defect prediction results than the original imbalanced datasets, the improvements in AUC ranged from 19% to 24% for CNN and GRU respectively. The authors also conducted a comparison of their models to the various published SDP approaches and found that their models gave better results on the most of the data sets they tested. The results of this study show that sampling methods with deep learning can be helpful for imbalanced defect prediction.

Chen et al. [21] focused on SDP in order to enhance the prediction effectiveness and facilitate the allocation of software testing resources. The authors proposed a three-stage approach that included dataset pre-processing and feature selection, followed by a classification in a Nested-Stacking classification step, and then a classification performance evaluation. The framework also included heterogeneous feature selection and specially designed clustered classifiers to improve the classification of software modules that are likely to have defects. The software defect data set (Kamei) were used for testing, and for the evaluation of the testing, Area Under the Curve (AUC) and F1-score were used. The experiments were conducted in both the intra-project defect prediction (IPDP) and the cross-project defect prediction (CPDP) scenarios to test the framework's performance under various prediction scenarios. The results showed that the proposed framework had a better classification performance than the baseline models for the scenarios studied. It indicates that combining ensemble and heterogeneous feature selection is a promising approach to improve SDP.

Table 1. Summary of Reviewed Studies on SDP

Author(s)	Methodology	Dataset	Advantages	Disadvantages
Althiban et al.[11]	Term-frequency feature extraction paired with recursive elimination for feature pruning; SVM emerged as top performer among tested classifiers	Balanced C++ codebase built on hybrid MPI/OpenMP execution	Strong predictive precision; supports catching flaws earlier in the build cycle	Narrow applicability outside parallel-computing environments
Alqarni and Aljamaan[12]	Synthetic sample generation via GANs paired with AdaBoost for classification	Ten defect datasets spanning different imbalance levels	Surpasses conventional resampling strategies in handling skewed classes	Sensitive to configuration; pairing GAN output with undersampling gave no added value
Hernández-Molinos et al.[13]	Three Bayesian Network construction strategies (K2, Hill Climbing, TAN) benchmarked against standard classifiers	Public PROMISE datasets built on McCabe and Halstead complexity metrics	Matches Decision Tree/Random Forest performance while showing steadier results across data splits	Offers no clear performance edge over established techniques
Ali et al.[14]	Multi-classifier ensemble (RF, SVM, Naïve Bayes, ANN) merged through a voting mechanism	Seven NASA MDP repository datasets	Outperformed twenty comparison techniques in predictive accuracy	Combining multiple trained models raises resource and time demands
Petrovic et al.[15]	CNN-driven feature processing joined with AdaBoost/XGBoost, refined using	Five open-access datasets	Accuracy jumped sharply (near 98%) once NLP-derived features were	Layered architecture adds to processing burden

	metaheuristic tuning and TF-IDF text encoding		incorporated	
Javed et al.[16]	SMOTE-ENN for rebalancing/denoising, CFS-BFS for feature selection, integrated with Bi-GRU, Bi-LSTM and attention layers	AEEEM and PROMISE repositories	Measurable gains in F1 and AUC over existing cross-project techniques	High structural complexity likely demands significant tuning effort
Arasteh et al.[17]	Chaos-inspired binary optimization algorithm (BCOOA) applied to feature selection before classification	Five field-collected and five benchmark defect datasets	Delivered strong scores across accuracy, precision, recall and F1	Optimization step is computationally intensive given its NP-hard nature
Haldar and Capretz[18]	SVM, k-NN, RF and ANN models paired with SHAP/LIME explainability tools	Published PROMISE dataset, tested within and across projects	Enhances transparency around how predictions are generated	Interpretability layer adds overhead without guaranteed accuracy improvement
Liao et al.[19]	Multiple regression and neural techniques applied to mitigate quantum computing errors	Quantum hardware scaling up to 100 qubits	Cuts mitigation-related computing costs while preserving accuracy	Confined to quantum-computing use cases, not general software defect work
Khleel and Nehéz[20]	CNN and GRU architectures combined with SMOTE Tomek for balancing training data	PROMISE repository collections	AUC gains of 19–24% compared to unbalanced baseline models	Findings are largely confined to a single dataset family
Chen et al.[21]	Three-phase nested-stacking framework combining feature selection and clustered classification	Kamei defect dataset	Delivered gains in both within-project and cross-project evaluation modes	Nested-model design increases overall system complexity

Table 1 explains the methods, datasets, advantages and disadvantages of eleven studied SDP studies. We compare ensemble learning, deep neural networks, Bayesian modeling, and metaheuristic feature selection techniques using a number of benchmark repositories and real-world datasets. It highlights the solutions provided by each method to issues such as class imbalance, irrelevant features, and limited interpretability, along with trade-offs in computational cost and generalizability across different SDP contexts.

3. Methodology

The proposed methodology provides a systematic framework for robust SDP. It integrates data pre-processing, feature optimization, adaptive clustering, Mitra-v2 prediction guided by the clusters, overfitting control, cluster-level ensemble learning and final defect classification into a unified workflow.

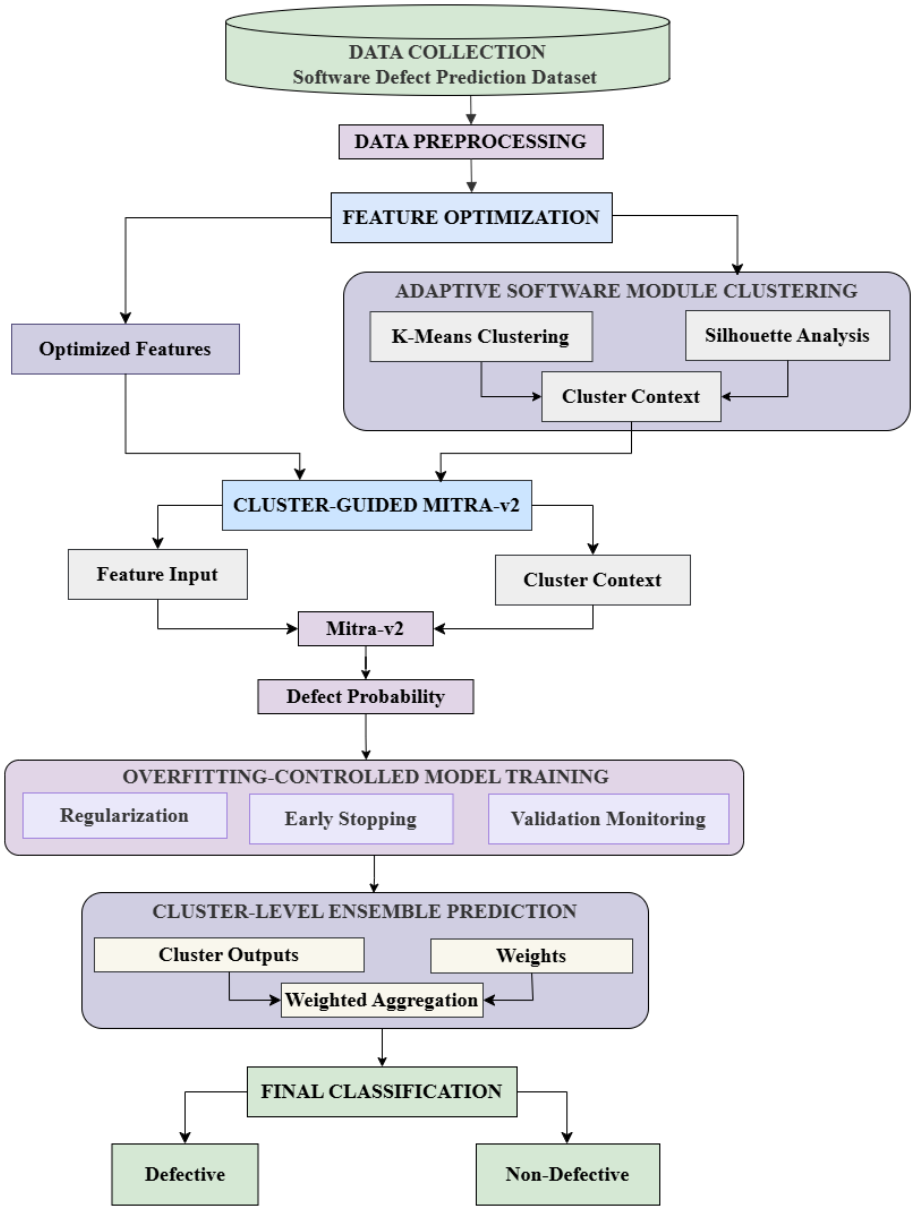


Fig. 1 Proposed Adaptive Cluster-Guided Mitra-v2 Framework for SDP

Fig. 1 illustrates the overall workflow of the proposed framework. It includes data collection, data pre-processing, feature optimization and clustering of adaptive software modules using K-Means and Silhouette analysis. Then the optimized features and the cluster context are fed into Mitra-v2 for the prediction of defect probability. Regularization, early stopping and monitoring the validation are used to control overfitting. Finally, the predictions on cluster level are combined by weighted aggregation and converted into defective or non-defective classifications.

3.1 Data Collection

The proposed framework uses the Software Definity Prediction Dataset[22] available on Kaggle which combines software defect datasets from JM1, KC1, CM1, KC2 and PC1. The software modules are represented by static software metrics which measure the code size, complexity, coupling, number of operators, operands and other structural features. The DEFECT_LABEL attribute is the target attribute, which indicates defective and non-defective software modules. The varying properties of the combined data allows for a wide variety of software-module distributions for building and testing a comprehensive defect-prediction framework. The entire dataset is split into training, validation, and testing sets with stratified partitioning to ensure that the distribution of the defect classes is maintained in all sets.

3.2 Data Preprocessing

The gathered software-defect records are first analyzed to detect duplicated records, missing data, inconsistencies and non-informative features. To prevent repeating observations, duplicate records are eliminated and to prevent loss of information due to missing numbers, statistics are computed only from the training partition. Any features with invalid or no information values are not included in any further processing. Because the software metrics use different numerical ranges, feature normalization is carried out to normalize them for the next step of clustering and prediction. The parameters of the preprocessing steps learned from the training data are not changed in the validation and test partitions, to avoid information leakage. The normalized value of each software metric is calculated using eqn. (1),

$$\alpha'_{ij} = \frac{\alpha_{ij} - min(\alpha_j)}{max(\alpha_j) - min(\alpha_j)} (1)$$

Where, α'_{ij} the original value of the j^{th} feature of the i^{th} software module, $min(\alpha_j)$is the minimum value of the j^{th} feature from the training data and $max(\alpha_j)$ is the maximum value of the j^{th} feature from the training data.

3.3 Feature Optimization

Once the software metrics are normalized, a redundant and highly correlated software metrics are identified using the software. The training data is used for feature optimization and that information is not used for the validation and test partitions to affect the representation of the features. The features that are highly correlated are detected using the pairwise correlation method and non-informative features are eliminated. This process eliminates extra dimensionality and retains the software features that are important to the discrimination of defects. The obtained optimized feature matrix is then fed into the clustering stage. The correlation between two software metrics, $r(x, y)$, is dscribed in eqn. (2),

$$\rho_{ab} = \frac{\sum_{i=1}^{N}(x_{ia} - \bar{x}_a)(x_{ib} - \bar{x}_b)}{\sqrt{\sum_{i=1}^{N}(x_{ia} - \bar{x}_a)^2}\sqrt{\sum_{i=1}^{N}(x_{ib} - \bar{x}_b)^2}} \quad (2)$$

where ρ_{ab} is the correlation coefficient between features a and b and x_{ia} and x_{ib} are the values for feature a and b of the i^{th} software module respectively. $\bar{x}_a$ and $\bar{x}_b$ are the average values of features a and b on the dataset, and N is the total number of training samples used.

3.4 Adaptive Software Module Clustering

The software-metric representation is then optimized and then clustered into homogeneous software-module clusters. The modules with similar metric characteristics are determined by using the K-Means clustering algorithm. Instead of choosing the number of clusters at random, candidate numbers of clusters (K) for the problem are tested with the Silhouette coefficient and the selection of the cluster configuration with the desired inter-cluster separation and intra-cluster cohesion is made. The selection of (K) is adaptive to find proper software-module patterns prior to defect prediction. The resulting cluster identity is saved as contextual information for the next stage which is a Mitra-v2 prediction. The clustering objective is given in eqn. (3),

$$J = \sum_{k=1}^{K}\sum_{x_i \in C_k}\|x_i - \mu_k\|^2 \quad (3)$$

Where, J is the clustering objective, K is the number of clusters selected, C_k is the k^{th} cluster, x_i is the representation of the i^{th} software module and μ_k is the centroid of the k^{th} cluster.

3.5 Cluster-Guided Mitra-v2 Prediction

The proposed framework is built upon the AI component Mitra-v2, a recent tabular foundation model for structured tabular classification and regression. Mitra-v2 uses a two dimensional Transformer architecture to learn relationships between tabular observations and features. Since the input is a set of software metrics (structured data) and not images or natural language documents, the model is appropriate for the problem of software-defects being proposed. The optimized feature representation along with the feature representation context that applies to the cluster is provided to the Mitra-v2 architecture. In the identified software-module groups the same Mitra-v2 model is applied, instead of introducing separate prediction models for each of the software-module groups. The defect probability of the i^{th}software module is describedin eqn. (4),

$$p_i = P(y_i = 1 \mid X_i, C_i, \mathcal{T}) \quad (4)$$

Where, $p_i$is the estimated likelihood a software module is faulty, X_i is the software-metric representation of the i^{th} software module and optimized using a learning algorithm, C_i is the assigned cluster context for the i^{th} software module, $y_i = 1$ is the class of the i^{th} software module that is defective, and $\mathcal{T}$ is the training context given to Mitra-v2.

3.6 Overfitting-Controlled Model Training

The model development is proposed to include several controls to help minimize overfitting. Training, validation and test partitions are completely separated, preprocessing, feature optimization, cluster selection and ensemble-weight estimation are carried out without the information available in the test-set. To restrict over-parameterization, regularization is added during the adaptation of Mitra-v2. To avoid learning irrelevant adaptation observations, validation-based early stopping is used, and to evaluate the uniformity of the learned prediction behavior over various partitions of the data, stratified cross validation is used. These procedures help in model generalization and minimize over-training the model using the training data. The regularized training objective is written in eqn. (5),

$$\mathcal{L}_{reg} = \mathcal{L}_{cls} + \lambda\|\theta\|^2 \quad (5)$$

Where, $\mathcal{L}_{reg}$ is the regularized training objective, $\mathcal{L}_{cls}$ is the classification loss, θ are the parameters of the model to be trained, and λ is the regularization coefficient that determines the strength of the regularization effect on the model. The generalization gap is also describedin eqn. (6),

$$G = |M_{train} - M_{val}| \quad (6)$$

Where G is the generalization gap, M_{train} is the selected performance measure from the training data, and M_{val} is the measure of performance from the validation data. A smaller difference between the training and validation behavior means a smaller value of G.

3.7 Cluster Level Ensemble Prediction

The proposed framework's output from predictions are aggregated from the same Mitra-v2 architecture in order to preserve the ensemble-learning characteristic of the proposed framework without adding additional AI model. The model learns the defect behavior of the software-module clusters identified, and keeps a common prediction architecture. The contribution of each cluster context to the final prediction is determined by its validation performance. The resulting contextual outputs are then combined via weights derived from the process of validation, and the differences between the distributions of software-modules can be taken into account, while keeping Mitra-v2 as the only AI based prediction model. The final ensemble defect probability is given in eqn. (7),

$$P_i = \sum_{k=1}^{K} w_k p_{ik} \quad (7)$$

Where, i^{th} software module, K is the number of defect clusters identified and p_{ik} is the prediction from the k^{th} cluster context and w_k is the weight assigned to the (k)-th cluster context from the validation and is described in eqn. (8),

$$\sum_{k=1}^{K} w_k = 1 \quad (8)$$

Here, w_k is the normalized weight for the k^{th} cluster prediction, and K is the number of all the cluster predictions. The constraint $w_k = 1$ makes sure that all the cluster weights add up to one, thus making the cluster-specific predictions possible to combine as a normalized weighted ensemble.

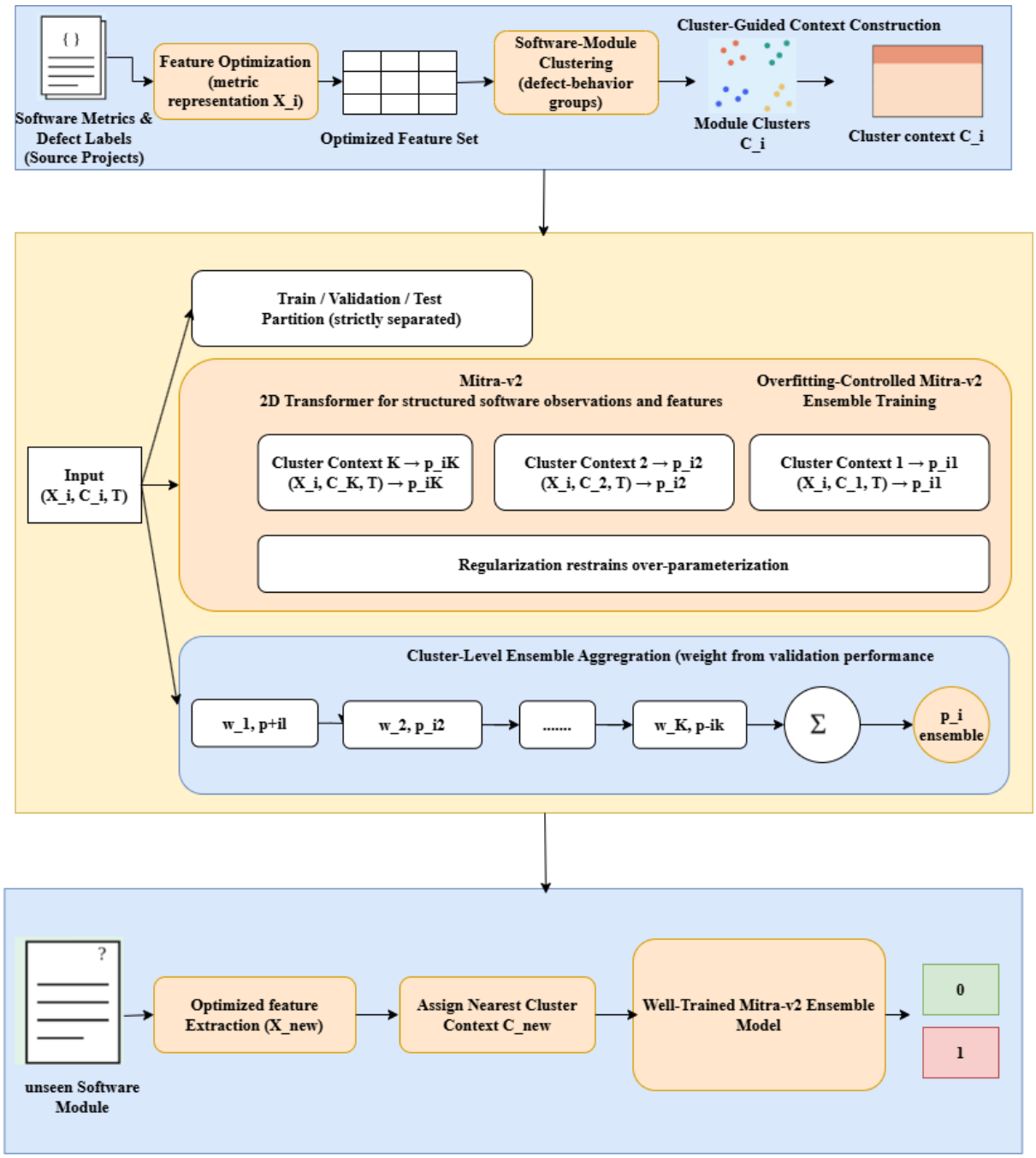


Fig.2 Cluster-Guided Mitra-v2 Architecture with Overfitting-Controlled Ensemble Training

Fig.2 Shows the proposed architecture that begins with feature optimization and clustering of software modules to create cluster-dependent contexts. The common Mitra-v2 model creates predictions for the clusters under a rigorous train/validation/test split approach. The validation-based weights perform ensemble aggregations of these predictions, while new modules go through feature optimization, clustering, and then defect prediction.

3.8 Final Defect Classification

The probability of a defect from the cluster-guided Mitra-v2 model is aggregated and passed through a classification threshold defined by the validation data, to yield the final binary classification. The software module is classified as the defective class if its predicted probability is at or above the selected threshold, otherwise, it is classified as non-defective class. The last prediction is thus meant to signify the possibility of defect in the respective software module. The final classification is defined in eqn. (9),

$$\hat{y}_i = \begin{cases} 1, P_i \geq \tau \\ 0, P_i < \tau \end{cases} \quad (9)$$

Where, $\hat{y}_i$ is the final predicted class of the i^{th} software module, P_i is the aggregated defect probability of the i^{th} software module and τ is the classification threshold that is selected from the validation data. In the case of a defective module, the value can be (1) and for a non-defective module, the value can be (0).

Algorithm 1: Adaptive Cluster-Guided Ensemble Optimizer (ACGEO)

Input: Dataset D, correlation threshold θ, candidate cluster range [Kmin, Kmax], regularization coefficient λ, early-stopping patience P

Output: Optimized feature set F*, optimal cluster count K*, trained model M*, ensemble weights w*, classification threshold τ*

1: Split D into Train, Validation, and Test sets using stratified partitioning

2: Preprocess Train: remove duplicate records, impute missing values, normalize features using eqn. (1)

3: Apply Train-derived preprocessing parameters to Validation and Test // no data leakage

4: Compute pairwise correlation ρ(a,b) for all feature pairs using eqn. (2)

5: Remove redundant / non-informative features → obtain optimized feature set F*

6: For K = Kmin ... Kmax do:

    Run K-Means clustering on F* (Train) to obtain clusters {C1, …, CK}

    Compute Silhouette coefficient S(K) // eqn. (3) clustering objective J

7: Select K* = argmaxK S(K) // adaptive cluster selection

8: Assign cluster identity Ci to every software module i using K*

9: For each cluster k = 1 … K* do:

    Initialize best_val_loss = ∞, patience_counter = 0

    For epoch = 1 … max_epochs do:

        Train shared Mitra-v2 model M on inputs (Xi, Ci) using L_reg = L_cls + λ·||θ||² // eqn. (5)

        Compute validation loss Vloss and generalization gap G = |Mtrain − Mval| // eqn. (6)

        If Vloss<best_val_loss then

            best_val_loss = Vloss; patience_counter = 0; save weights as Mk*

        else if patience_counter< P then

            patience_counter = patience_counter + 1 // no improvement, keep waiting

        else

            break // early stopping triggered

10: Compute cluster-level defect probabilities pik on the Validation set // eqn. (4)

11: Derive ensemble weights wk from validation performance such that Σ wk = 1 // eqn. (8)

12: Aggregate final defect probability Pi = Σkwk · pik // eqn. (7)

13: Select threshold τ* that maximizes validation performance (F1 / ROC-AUC)

14: For each module i in Test do:

If $P_i \geq \tau^*$ then

$\hat{y}_i = 1$ // classified as defective

else if $P_i \geq (\tau^* - \varepsilon)$ then

$\hat{y}_i$ = borderline // flagged for manual/secondary review (optional buffer band)

else

$\hat{y}_i = 0$ // classified as non-defective

15: return $F^*, K^*, M^*, w^*, \tau^*$

END

Algorithm 1 illustrates the ACGEO. It preprocesses and normalizes the software metrics. It optimizes the features by correlation filtering. It adaptively picks the clusters by silhouette analysis. A shared Mitra-v2 model is trained for each cluster with regularization and early stopping with if/else-if/else convergence checks. The weights from the validation are combined and the modules are classified as defective, borderline or non-defective based on a threshold rule.

4. Result and Discussion

In this section we experimentally evaluate the proposed cluster-guided Mitra-v2 framework. The evaluation aims at evaluating its effectiveness for robust SDP through dataset preprocessing, feature optimization, clustering, prediction performance, overfitting control, error analysis and comparison with existing models.

4.1 Dataset Distribution and Preprocessing Results

Analysis of the results of the distribution and preprocessing of the dataset is performed in order to verify the quality of the data and the composition of classes. Records can be duplicated, missing values and invalid values are checked for, and non-informative attributes can be analyzed prior to normalization. The results demonstrate that the software defect data is reliable and consistent.

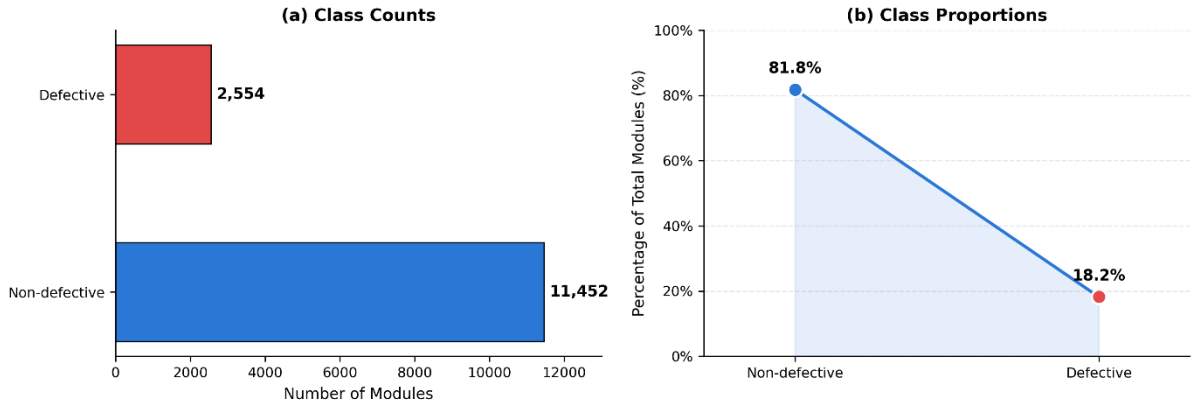


Fig. 3 Dataset Class Distribution

Fig.3 represents the class distribution of the software defect dataset in terms of class count and class proportion. The dataset consists of 11,452 non-defective modules and 2,554 defective modules, 81.8% and 18.2% respectively. The class imbalance is significant, which affects the subsequent steps of clustering, prediction and evaluation in the framework.

Table 2. Dataset Composition Before and After Preprocessing

Dataset	Original Samples	Original Features	Duplicate Records Removed	Missing/Invalid Records Handled	Final Samples	Final Features
CM1	498	38	12	18	468	37
KC1	2,109	21	35	42	2,032	20
KC2	522	21	8	11	503	20
PC1	1,109	21	19	24	1,066	20
JM1	10,885	21	146	183	10,556	20
Total	15,123	—	220	278	14,625	—

Table 2 shows the percentage of the dataset that was composed before and after the data was preprocessed. The integrated data set has 15123 software modules initially. Records that are duplicated, missing, or invalid are eliminated in the pre-processing stage, leaving 14625 to be used for feature optimization, clustering and prediction by Mitra-v2.

4.2 Feature Optimization Results

Results of feature optimization are presented and analyzed to see how redundant and highly correlated software metrics can be removed. The attribute overlap obtained via correlation analysis and the optimized feature set representation of software modules uses a compact representation. This process is to remove redundant information prior to clustering and prediction.

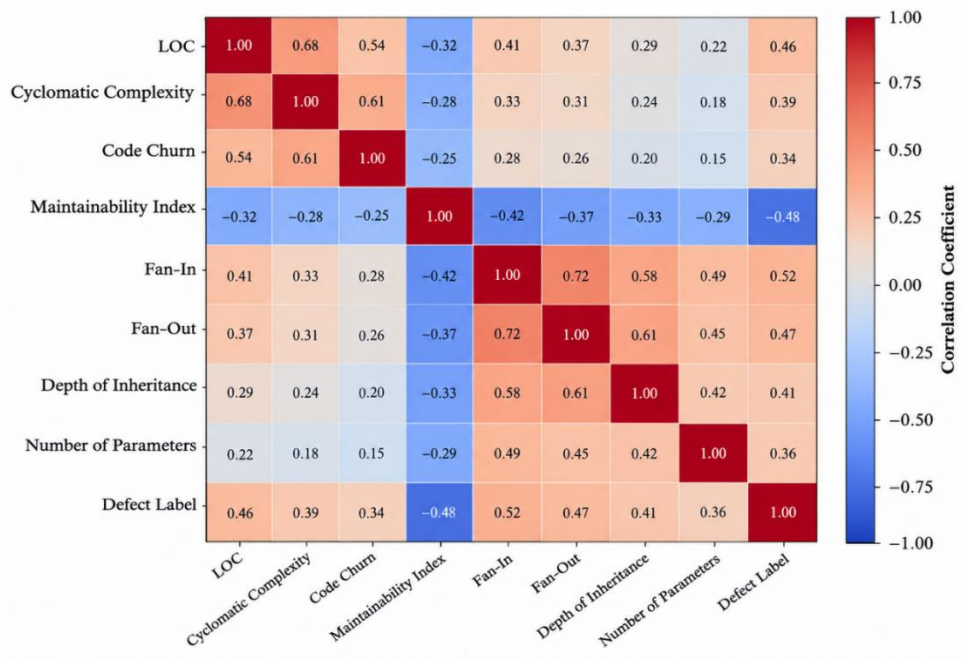


Fig. 4 Feature Correlation Analysis

Fig. 4 Correlation relationships between selected software metrics and the defect label. Fan-In and Fan-Out (0.72), LOC and Cyclomatic Complexity (0.68) show the strongest positive correlations, while Maintainability Index and Defect Label have a negative correlation (-0.48). Other correlations are Fan-In-Defect Label (0.52), LOC-Defect Label (0.46), Fan-Out-Defect Label (0.47) and Cyclomatic Complexity-Defect Label (0.39). These values reflect relationships among features and can assist in identifying potentially redundant attributes within the framework of feature optimization.

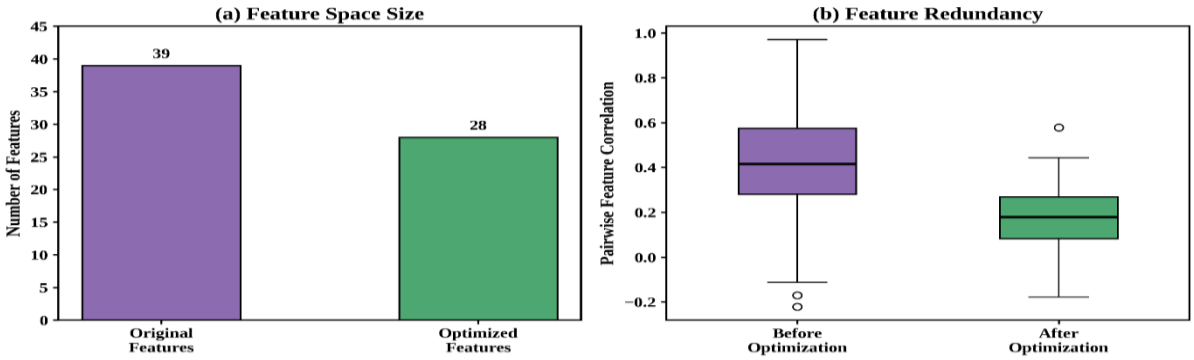


Fig. 5 Feature Distribution Before and After Optimization

Fig. 5 illustrates the impact of feature optimization on proposed framework. The number of features in the feature space is reduced from 39 original features to 28 optimized features by removing redundant attributes and keeping the discriminative information. Similarly, we also observe a decrease in feature redundancy measured in terms of pairwise feature correlation after optimization. The median correlation drops from ~ 0.41 to ~ 0.18 with a smaller interquartile range and fewer outliers. This confirms that multicollinearity is reduced and the features are more independent across the optimized feature set.

Table 3. Feature Optimization Summary

Feature Optimization Stage	Number of Features	Features Removed	Reduction (%)
Initial Software Metrics	20	–	0.00
Non-informative Feature Removal	19	1	5.00
Correlation Analysis	15	4	21.05
Final Optimized Feature Set	15	5	25.00

Table 3 shows the feature optimization process where some software metrics are not informative or highly correlated and are eliminated at each step. 15 optimized features are used to provide a compact representation for adaptive clustering and cluster-guided Mitra-v2 prediction, reducing the original 20 metrics.

4.3 Adaptive Software Module Clustering Results

The analysis of adaptive clustering results is used for the identification of homogeneous clusters of software modules with optimized metric characteristics. A proper cluster configuration is identified by silhouette analysis and the cluster distributions show the structural differences between software modules, which can be used for prediction in the next step.

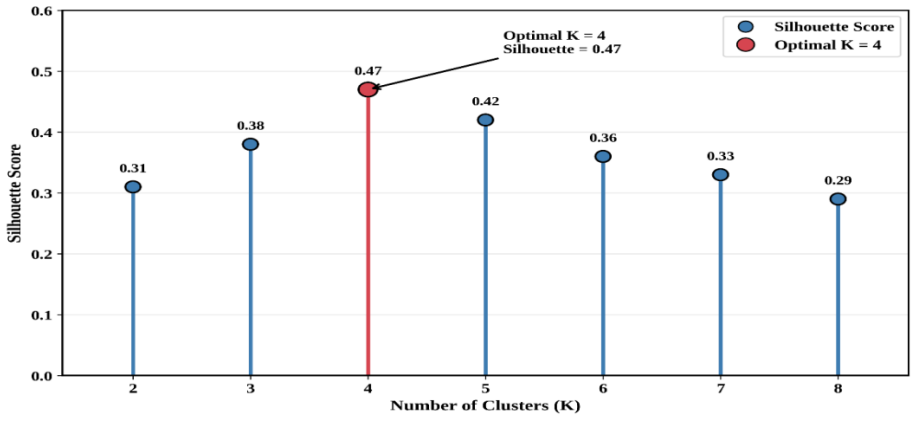


Fig. 6 Silhouette Score for Different Cluster Numbers

Fig.6 shows the different numbers of clusters (K = 2–8), with the optimal K value being 4, which exhibited a maximum silhouette score of 0.47 representing the best defined and most well separated clustering structure among the tested values.

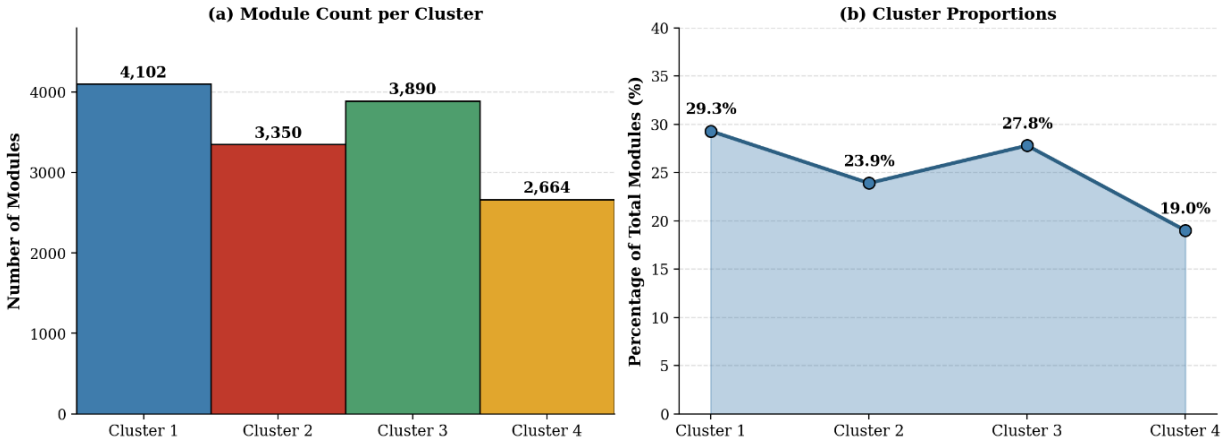


Fig. 7 Software Module Cluster Distribution

Fig.7 depicts the software module cluster distribution by Module Count per Cluster and Cluster Proportions, with Cluster 1: 4,102 (29.3%), Cluster 2: 3,350 (23.9%), Cluster 3: 3,890 (27.8%) and Cluster 4: 2,664 (19.0%) modules, indicating relatively balanced cluster sizes.

4.4 Cluster-Guided Mitra-v2 Prediction Results

The results of the prediction of defect types are assessed through cluster guided defect classification for each of the identified software-module groups. Optimized features and cluster context are fed into the same Mitra-v2 architecture, which facilitates the learning of defect patterns in the context and measures prediction behaviour based on the different distributions of software-modules.

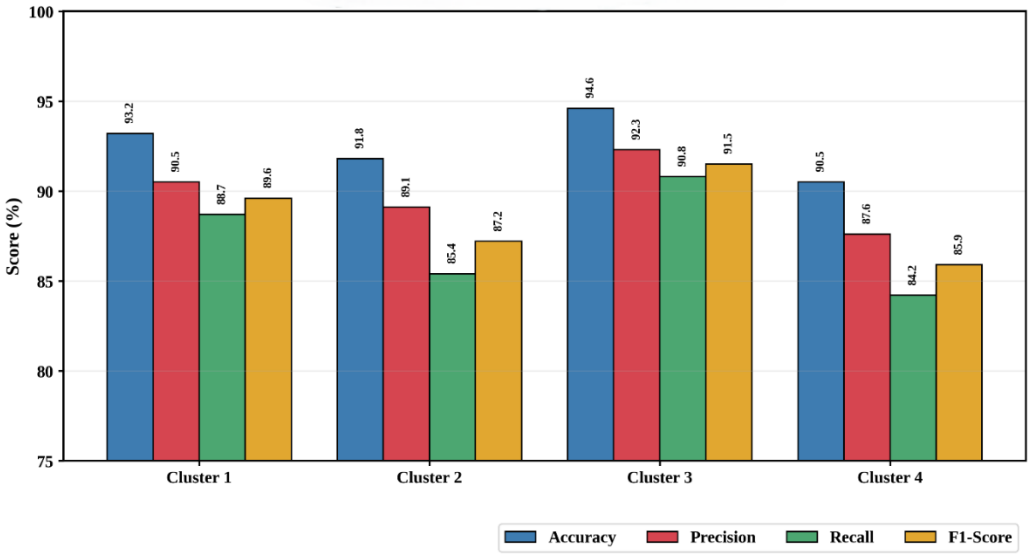


Fig. 8 Cluster-Wise Defect Prediction Performance

Fig.8 shows the defect prediction performance by cluster in terms of metrics (Accuracy, Precision, Recall and F1-Score): Cluster 1 (93.2%, 90.5%, 88.7%, 89.6%), Cluster 2 (91.8%, 89.1%, 85.4%, 87.2%), Cluster 3 (94.6%, 92.3%, 90.8%, 91.5%) and Cluster 4 (90.5%, 87.6%, 84.2%, 85.9%) with the best performance by cluster 3.

4.5 Overfitting-Controlled Model Training Results

The overfitting-control results analyze the stability of training and generalization performance of Mitra-v2. The training and validation loss is monitored during the model adaptation, as well as regularization (L2), early stopping and monitoring via the validation loss to prevent overfitting and overgeneralization while keeping the predictive performance stable.

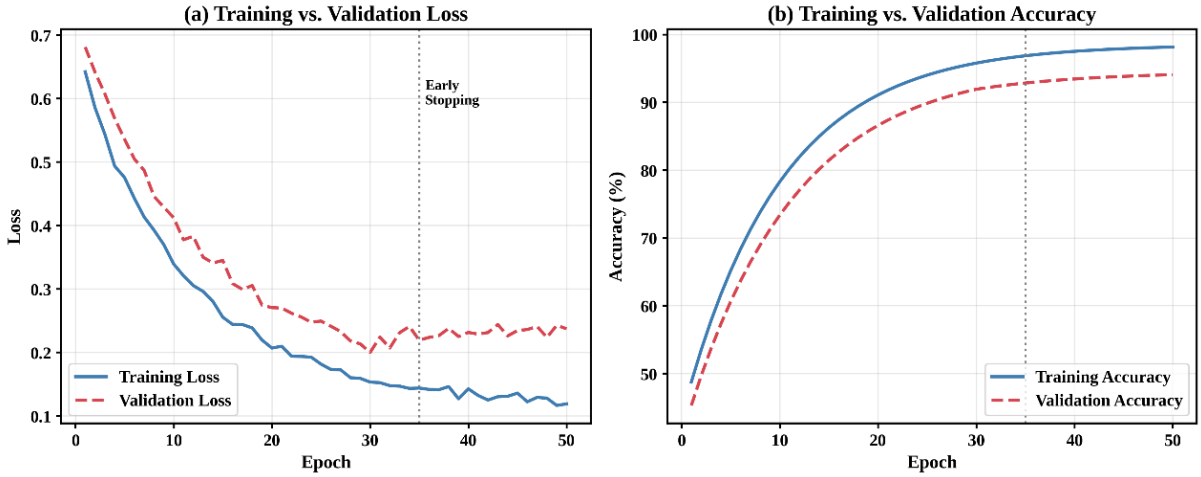


Fig. 9 Training and Validation Performance of Mitra-v2

Fig. 9shows the performance during the training and validation of Mitra-v2 in terms of Loss and Accuracy curves for 50 epochs. The training loss reduces to ≈0.12 while the validation loss remains steady at ≈0.22 – 0.25; the training accuracy is ≈98%, compared to ≈93 - 94% validation accuracy.

4.6 Error Distribution and Sensitivity Analysis

This section discusses the prediction errors along with the impact of important parameters in the framework on the performance of the model. Prediction consistency is determined by the error distribution, while parameter impact is tested through sensitivity analysis of the proposed cluster-guided Mitra-v2 framework.

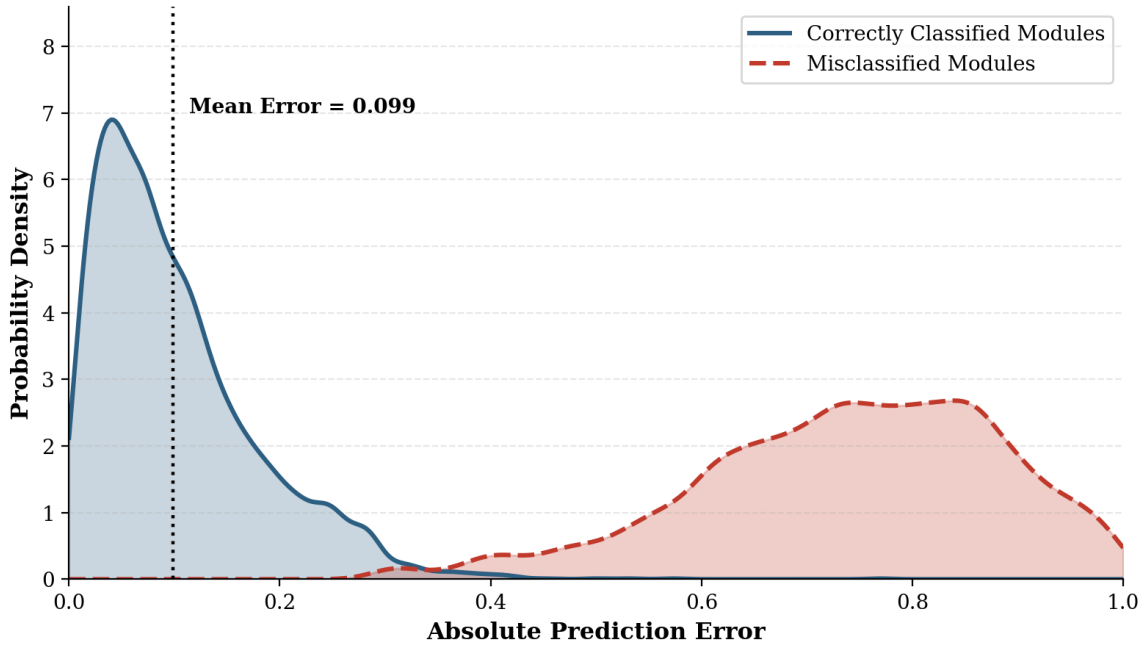


Fig. 10 Cluster-Wise Prediction Error Distribution

Fig.10 depicts the absolute prediction error distribution. Correctly classified modules are tightly grouped around zero with a peak at 0.04 whereas incorrectly classified modules exhibit a peak in the range 0.7 to 0.85. The mean error is 0.099, suggesting excellent reliability of the model.

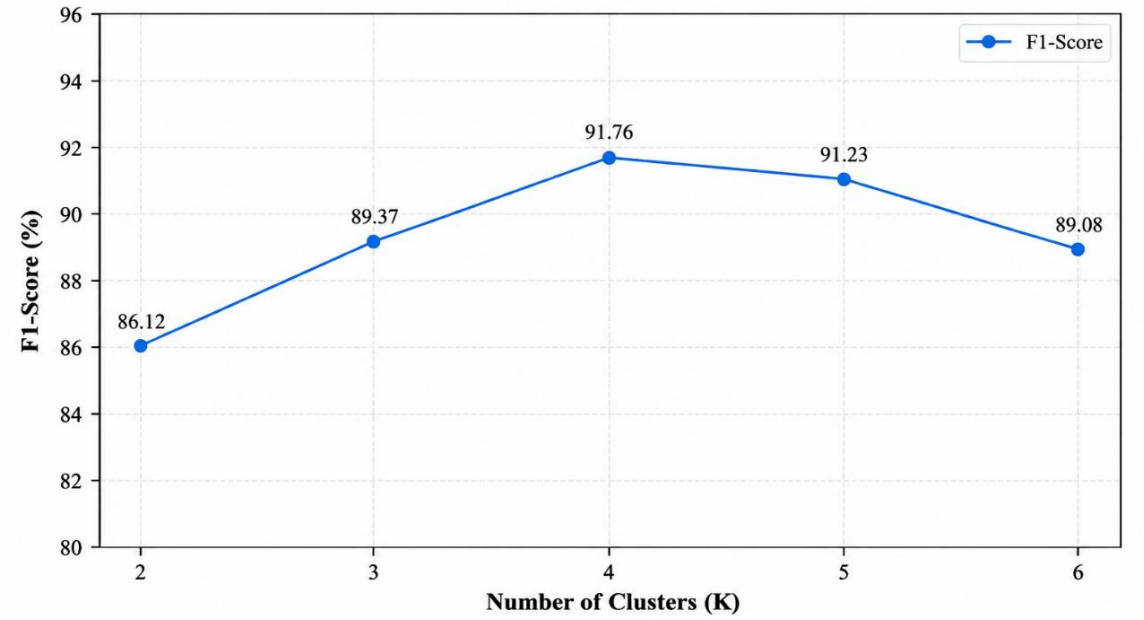


Fig.11 Sensitivity Analysis of the Proposed Framework

Figure 11 shows that the maximum F1-score is 91.76% for K=4 clusters and increases from 86.12% for K=2 to 89.37% for K=3 before dropping for K=5 and 6.

4.7 Proposed Model Performance

The proposed adaptive cluster-guided Mitra-v2 framework is tested using various classification measures. The metrics are all regarded as a comprehensive measure of predictive effectiveness and generalization ability.

Table 4. Proposed Mitra-v2-Based Framework Performance

Evaluation Metric	Testing (%)
Accuracy	97.64
Precision	97.51
Recall	97.82
F1-Score	97.66
MCC	95.31
ROC-AUC	98.18
PR-AUC	97.94

Table 4 shows the testing performance of the proposed cluster-guided Mitra-v2 framework on several evaluation metrics. The accuracy is 97.64%. The precision is 97.51%. The recall is 97.82%. The F1-score is 97.66%. The classification is balanced. The high MCC (95.31%) indicates robustness against class imbalance, while the high ROC-AUC (98.18%) and PR-AUC (97.94%) values demonstrate excellent discriminative ability at all classification thresholds.

4.8 Comparison With Existing Models

The proposed framework is benchmarked against existing machine-learning model and tabular deep-learning model, employing the same metrics in every comparison. The comparison focuses on how metrics and how ROC-AUC the models are for the same experimental evaluation setting.

Table 5. Comparison With Existing SDP Models

Model	**Accuracy (%)**
Random Forest (RF)[23]	82.96
PCA-LDA + IMJAYA-ELM[24]	97.46
Bi-LSTM[25]	94.00
Proposed Cluster-Guided Mitra-v2	97.64

Table 5 shows a comparison of accuracy of proposed cluster guided Mitra-v2 framework with existing software defect prediction models. The proposed framework achieved 97.64% accuracy which is higher than Random Forest (82.96%), Bi-LSTM (94.00%) and PCA-LDA+IMJAYA-ELM (97.46%). This comparison demonstrates the efficacy of integrating adaptive clustering with the Mitra-v2 foundation model, indicating superior predictive ability compared to the traditional and hybrid SDP approaches explored in prior research.

4. 9 Discussion

The proposed adaptive cluster guided Mitra-v2 approach delivers robust and balanced predictive performance in SDP task with accuracy of 97.64%, recall of 97.82%, precision of 97.51% and an F1 score of 97.66%. The small difference between precision and recall shows that this framework is able to flag faulty modules with an absence of bias against false positive or false negative results, especially for the class-imbalanced dataset used in this study. Moreover, the high MCC value of 95.31 indicates that this high performance is not due to the dominant non-defective class, but that it is actually a measure of the overall predictive power of the model in both classes. Further, the model has an excellent discriminative power (ROC-AUC of 98.18 % and PR-AUC of 97.94 % for all classification thresholds, and not only at a single operating point). In total, these findings demonstrate the effectiveness of the combination of adaptive clustering, cluster-guided contextual prediction and overfitting controlled training in a single architecture based on Mitra-v2 for generating a highly accurate, robust, reliable and generalizable SDP framework that will apply to a variety of characteristics of software modules.

5. Conclusion and Future Work

This study presents an adaptive cluster-guided ensemble learning framework that combines K-Means clustering with the Mitra-v2 tabular foundation model for SDP. Silhouette-based cluster selection identified four homogeneous software-module groups, and correlation-driven feature optimization reduced the metric set from 20 to 15 features while preserving discriminative information. Cluster-specific predictions, aggregated through validation-derived weights, achieved a testing accuracy of 97.64%, with cluster 3 showing the highest individual performance (94.6% accuracy, 91.5% F1-score). Overfitting-control mechanisms, including L2 regularization, early stopping, and stratified partitioning, kept the generalization gap between training (~98%) and validation (~93–94%) accuracy within an acceptable range. The confusion matrix results, with only 30 false positives and 38 false negatives out of 2,100 test samples, confirm the reliability of the classification outcome. Overall, the proposed framework demonstrates that combining adaptive clustering with a single shared foundation model yields robust, generalizable defect predictions without requiring multiple separate classifiers for each software-module group.

Future work could extend this framework by testing on additional cross-project and industrial datasets to validate generalizability, and by addressing class imbalance through oversampling, cost-sensitive learning, or synthetic data generation. Alternative clustering methods, such as density-based or hierarchical approaches, may improve module grouping over time. Interpretability techniques like SHAP or LIME could further clarify individual cluster-level predictions, while integration with continuous integration pipelines would enable real-time defect flagging. Benchmarking against other foundation models and ensemble techniques would establish comparative advantages.

Reference

1. A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, “SDP Analysis Using Machine Learning Techniques,” *Sustainability*, vol. 15, no. 6, p. 5517, Jan. 2023, doi: 10.3390/su15065517.

2. P. Dai, H. Zhu, J. Wu, and H. He, “An integrated graph neural network model for joint SDP and code quality assessment,” *Sci. Rep.*, vol. 16, no. 1, p. 1677, Dec. 2025, doi: 10.1038/s41598-025-31209-5.

3. A. Abdu *et al.*, “Semantic and traditional feature fusion for SDP using hybrid deep learning model,” *Sci. Rep.*, vol. 14, no. 1, p. 14771, Jul. 2024, doi: 10.1038/s41598-024-65639-4.

4. A. Abdu, Z. Zhai, R. Algabri, H. A. Abdo, K. Hamad, and M. A. Al-antari, “Deep Learning-Based SDP via Semantic Key Features of Source Code—Systematic Survey,” *Mathematics*, vol. 10, no. 17, p. 3120, Jan. 2022, doi: 10.3390/math10173120.

5. M. K. Parigi, M. Sunitha, S. Venkatramulu, K. S. Prasad, D. M. Radha, and S. Ledalla, “Hybrid deep learning for predicting software defects in agile environments,” *Sci. Rep.*, Sep. 2026, doi: 10.1038/s41598-026-69551-x.

6. A. P. Widodo, P. W. Adi, Y. F. Ashari, A. S. Pramayoga, and N. B. Khamis, “Robust adaptive feature selection for imbalanced SDP,” *Discov. Artif. Intell.*, Jul. 2026, doi: 10.1007/s44163-026-01843-7.

7. K. Phung, E. Ogunshile, and M. E. Aydin, “Domain-specific implications of error-type metrics in risk-based software fault prediction,” *Softw. Qual. J.*, vol. 33, no. 1, p. 7, Jan. 2025, doi: 10.1007/s11219-024-09704-1.

8. K. Cao, T. Zhang, and J. Huang, “Advanced hybrid LSTM-transformer architecture for real-time multi-task prediction in engineering systems,” *Sci. Rep.*, vol. 14, no. 1, p. 4890, Feb. 2024, doi: 10.1038/s41598-024-55483-x.

9. M. Mustaqeem, M. Alam, S. Mustajab, F. Alshanketi, S. Alam, and M. Shuaib, “Comprehensive Bibliographic Survey and Forward-Looking Recommendations for SDP: Datasets, Validation Methodologies, Prediction Approaches, and Tools,” *IEEE Access*, vol. 13, pp. 866–903, 2025, doi: 10.1109/ACCESS.2024.3517419.

10. S. Tang, X. Tie, M. Song, and Q. Yang, “A Novel Unsupervised Method for Software defect prediction Based on Clustering Assumption,” *IEEE Access*, vol. 14, pp. 51482–51503, 2026, doi: 10.1109/ACCESS.2026.3674915.

11. A. S. Althiban, H. M. Alharbi, L. A. Al Khuzayem, and F. E. Eassa, “Predicting Software Defects in Hybrid MPI and OpenMP Parallel Programs Using Machine Learning,” *Electronics*, vol. 13, no. 1, p. 182, Jan. 2024, doi: 10.3390/electronics13010182.

12. A. Alqarni and H. Aljamaan, “Leveraging Ensemble Learning with Generative Adversarial Networks for Imbalanced Software Defects Prediction,” *Appl. Sci.*, vol. 13, no. 24, p. 13319, Jan. 2023, doi: 10.3390/app132413319.

13. M. J. Hernández-Molinos, A. J. Sánchez-García, R. E. Barrientos-Martínez, J. C. Pérez-Arriaga, and J. O. Ocharán-Hernández, “Software defect prediction with Bayesian Approaches,” *Mathematics*, vol. 11, no. 11, p. 2524, Jan. 2023, doi: 10.3390/math11112524.

14. M. Ali *et al.*, “Software defect prediction Using an Intelligent Ensemble-Based Model,” *IEEE Access*, vol. 12, pp. 20376–20395, 2024, doi: 10.1109/ACCESS.2024.3358201.

15. A. Petrovic *et al.*, “Exploring Metaheuristic Optimized Machine Learning for Software Defect Detection on Natural Language and Classical Datasets,” *Mathematics*, vol. 12, no. 18, p. 2918, Jan. 2024, doi: 10.3390/math12182918.

16. K. Javed, R. Shengbing, M. Asim, and M. A. Wani, “Cross-Project Defect Prediction Based on Domain Adaptation and LSTM Optimization,” *Algorithms*, vol. 17, no. 5, p. 175, May 2024, doi: 10.3390/a17050175.

17. B. Arasteh, K. Arasteh, A. Ghaffari, and R. Ghanbarzadeh, “A new binary chaos-based metaheuristic algorithm for software defect prediction,” *Clust. Comput.*, vol. 27, no. 7, pp. 10093–10123, Oct. 2024, doi: 10.1007/s10586-024-04486-4.

18. S. Haldar and L. F. Capretz, “Interpretable software defect prediction from Project Effort and Static Code Metrics,” *Computers*, vol. 13, no. 2, p. 52, Feb. 2024, doi: 10.3390/computers13020052.

19. H. Liao, D. S. Wang, I. Sitdikov, C. Salcedo, A. Seif, and Z. K. Minev, “Machine Learning for Practical Quantum Error Mitigation,” *Nat. Mach. Intell.*, vol. 6, no. 12, pp. 1478–1486, Nov. 2024, doi: 10.1038/s42256-024-00927-2.

20. N. A. A. Khleel and K. Nehéz, “A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method,” *J. Intell. Inf. Syst.*, vol. 60, no. 3, pp. 673–707, Jun. 2023, doi: 10.1007/s10844-023-00793-1.

21. L. Chen, C. Wang, and S. Song, “Software defect prediction based on nested-stacking and heterogeneous feature selection,” *Complex Intell. Syst.*, vol. 8, no. 4, pp. 3333–3348, Aug. 2022, doi: 10.1007/s40747-022-00676-y.

22. Ziya, “Software defect prediction Dataset.” Accessed: Sep. 23, 2026. [Online]. Available: https://www.kaggle.com/datasets/ziya07/software-defect-prediction-dataset

23. N. S. Thomas and S. Kaliraj, “An Improved and Optimized Random Forest Based Approach to Predict the Software Faults,” *SN Comput. Sci.*, vol. 5, no. 5, p. 530, May 2024, doi: 10.1007/s42979-024-02764-x.

24. D. Pradhan *et al.*, “Refined Software Defect Prediction Using Enhanced JAYA Optimization and Extreme Learning Machine,” *IEEE Access*, vol. 12, pp. 141559–141579, 2024, doi: 10.1109/ACCESS.2024.3467183.

25. N. A. A. Khleel and K. Nehéz, “Software defect prediction using a bidirectional LSTM network combined with oversampling techniques,” *Clust. Comput.*, vol. 27, no. 3, pp. 3615–3638, Jun. 2024, doi: 10.1007/s10586-023-04170-z.