DOI: http://dx.doi.org/10.26483/ijarcs.v16i1.7192

Volume 16, No. 1, January-February 2025

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

MACHINE LEARNING BASED HYBRID APPROACH FOR SOFTWARE DEFECT PREDICTION

Adarsh
Research Scholar, NIILM University,
Kaithal, Haryana India

Pawan Kumar
Assoc. Professor, NIILM University,
Kaithal, Haryana India

Abstract: Ensuring software reliability is a critical challenge in the tech industry, traditionally addressed through manual inspection and experience-based techniques that can be time-consuming and inefficient. Automated software defect prediction models, leveraging machine learning, offer a proactive solution to identify and mitigate defects early in the development cycle. This study proposes a defect prediction model based on hybrid learning techniques, evaluating its performance against Decision Trees (DT), Naïve Bayes (NB), Artificial Neural Networks (ANN), and Support Vector Machines (SVM). Using standard evaluation metrics such as ten-fold cross-validation, precision, recall, specificity, F1-score, and accuracy, our findings demonstrate that hybrid learning consistently outperforms other models, achieving classification accuracy between 98% and 100% across multiple datasets (JM1, CM1, and PC1). While DT also performs well, NB and ANN require careful tuning, and SVM exhibits the lowest accuracy. These results highlight hybrid learning as a robust and effective approach for enhancing software reliability by improving defect prediction accuracy.

Keywords: Software defect, Machine Learning Hybrid Learning, SVM, Decision Tree.

1. INTRODUCTION

Software defect prediction has long been a critical area of research in software engineering, aiming to improve software reliability and reduce development costs. Traditionally, defect prediction relied on manual inspection and expert-driven techniques, which, while useful, were often time-consuming, error-prone, and difficult to scale. These early methods primarily depended on historical data, heuristic rules, and domain expertise, making them limited in their ability to adapt to new and evolving software environments. Furthermore, as software systems grew in complexity, traditional approaches struggled to handle large-scale projects effectively, often leading to inaccurate predictions and undetected defects [1-2].

With the advancement of machine learning (ML), automated software defect prediction has emerged as a promising solution to overcome these limitations. Machine learning techniques can analyze vast amounts of data from diverse sources, including code repositories, bug tracking systems, and testing frameworks, to identify patterns and predict potential defects with high accuracy. Among various ML approaches, hybrid learning has gained significant attention due to its ability to combine multiple models and improve predictive performance. Unlike single classifier models, which may suffer from high dimensionality and imbalanced datasets, hybrid learning techniques integrate multiple classifiers to enhance robustness and generalization [3-4].

In this study, we propose a machine learning-based defect prediction model leveraging hybrid learning techniques to enhance accuracy and reliability. Our model is evaluated against other widely used machine learning algorithms, including Decision Trees (DT) [5], Naïve Bayes (NB) [6], Artificial Neural Networks (ANN) [7], and Support Vector Machines (SVM) [8]. The evaluation is conducted using standard performance metrics such as precision, recall, specificity, F1-score, and accuracy, along with ten-fold cross-validation to ensure reliable results [9-10]. Experimental results indicate that hybrid learning consistently outperforms other models, achieving classification accuracy between 98% and 100% across multiple datasets (JM1, CM1, and PC1) [11].

The findings of this research highlight the importance of adopting machine learning-based automated defect prediction models in modern software development. By integrating these models into software development pipelines, particularly in DevOps and CI/CD environments, organizations can proactively identify and mitigate defects, ultimately enhancing software quality, reducing maintenance costs, and improving overall software reliability. Additionally, our study underscores the need for further research in deep learning and explainable AI (XAI) techniques to make defect prediction models more interpretable and adaptable to evolving software environments [12]. As software complexity continues to grow, the adoption of intelligent, automated defect prediction models will become increasingly essential in ensuring efficient and high-quality software development.

2. RELATED WORK

There are various algorithms and methodologies applied to software defect prediction (SDP), each with distinct features and advantages. Case-based reasoning (CBR) [1] is effective for distributed software applications, improving fault prediction in complex systems. Another study explores feature subset selection and classification techniques [2], demonstrating improved cross-project defect prediction (CPDP) through refined feature selection methods. The Automatic Learning and Fault Tolerance Approach [3-4] learns user expectations from semantic contexts, making it adaptable to different software applications.

A comparison of Random Forest (RF), Support Vector Machines (SVM), C4.5 (Decision Tree), and Regression Tree [5] indicates that RF achieves the highest accuracy, making it a strong choice for defect prediction. Kernel Principal Component Analysis (KPCA) and Weighted Extreme Learning Machine (WELM) [6] applied to data from 44

software projects show better results than baseline methods. Artificial Neural Networks (ANN), SVM, Decision Trees (DT), CCN, GMDH, and GEP [7] were analyzed, with DT demonstrating the highest accuracy. Soft Computing-based Machine Learning [8] optimizes feature prediction in high-cost, safety-critical industries.

Unsupervised learning methods such as FCM and FSOM [9] show comparable performance to supervised models in a meta-analysis of 2,456 results, though reporting concerns remain. Hybrid Learning techniques [10], including Random Forest, Boosting, and Bagging, have advanced significantly, particularly with effective feature selection and data sampling. Deep learning approaches, including CNNs [11, 14], are recommended for SDP, particularly in feature extraction and handling class imbalance, while ANNs [12] are noted for their role in defect prediction trends. Support Vector Machines (SVM) [13] introduce novel filtering techniques to improve accuracy, AUC, and F-measure in SDP. General machine learning approaches [15-16] have been analyzed for their business-driven potential in software defect prediction, identifying key trends and opportunities for adoption.

3. MACHINE LEARNING ALGORITHMS

The study evaluates and compares three supervised machine learning algorithms: Naïve Bayes (NB), Artificial Neural Networks (ANN), and Decision Trees (DT), focusing on their accuracy and effectiveness in predicting software defects.

Naïve Bayes (NB): A probabilistic classifier based on Bayes' theorem, Naïve Bayes assumes feature independence, allowing for efficient computation and strong performance in large datasets with high-dimensional data. Despite its simplicity, NB is effective in various real-world applications, making it a valuable tool for defect prediction [13].

Artificial Neural Networks (ANNs): Inspired by biological neural networks, ANNs are powerful non-linear classifiers capable of modeling complex input-output relationships. Comprising interconnected processing units called neurons, ANNs learn from data and adapt to patterns, making them effective for predictive analytics, image recognition, and natural language processing [14].

Support Vector Machines (SVM): SVMs are supervised learning models designed for classification and regression tasks. They function by identifying the optimal hyperplane that best separates data classes, maximizing the margin between support vectors. SVMs perform well in high-dimensional spaces and effectively handle both linear and non-linear data through kernel functions [15].

Decision Trees (DT): A widely used method in machine learning and data mining, Decision Trees provide interpretable, hierarchical models that classify data points based on a series of decision rules. Each node in the tree represents a decision based on a feature value, leading to an outcome at the leaf nodes. While highly intuitive and easy to visualize, Decision Trees can be prone to overfitting, but techniques like pruning help improve generalization [16].

Hybrid Learning (HL): Hybrid learning is a powerful machine learning technique that combines multiple individual models to improve overall predictive performance and robustness. Rather than relying on a single model, hybrid methods aggregate the predictions of several models, often leading to better accuracy and generalization. Common hybrid techniques include bagging, boosting, and stacking. Bagging, such as Random Forest, involves training multiple models on different subsets of the data and averaging their predictions to reduce variance. Boosting, on the other hand, sequentially trains models, where each new model focuses on correcting the errors of the previous ones, thus reducing bias. Stacking involves training a meta-model to combine the predictions of various base models optimally. Hybrid learning is particularly effective in complex, high-dimensional datasets, as it leverages the strengths of various models while mitigating their individual weaknesses, resulting in a more robust and reliable predictive system.

4. PROPOSED HYBRID APPROACH FOR SDP

The proposed hybrid approach for software defect prediction integrates multiple machine learning models to maximize accuracy and generalization. This framework consists of three primary stages: feature selection, model training, and hybrid integration. Initially, feature selection techniques such as Principal Component Analysis (PCA) and Recursive Feature Elimination (RFE) are applied to reduce dimensionality and enhance model interpretability. Next, a combination of base classifiers—including Naïve Bayes, Decision Trees, and Support Vector Machines—is trained on the processed dataset. The predictions from these models are then combined using a meta-learning strategy, such as stacking, where a higher-level model learns to optimally weight each base model's contribution. By integrating diverse learning paradigms, the proposed approach aims to balance bias-variance trade-offs, improve classification accuracy, and ensure robustness in defect prediction across varying software datasets.

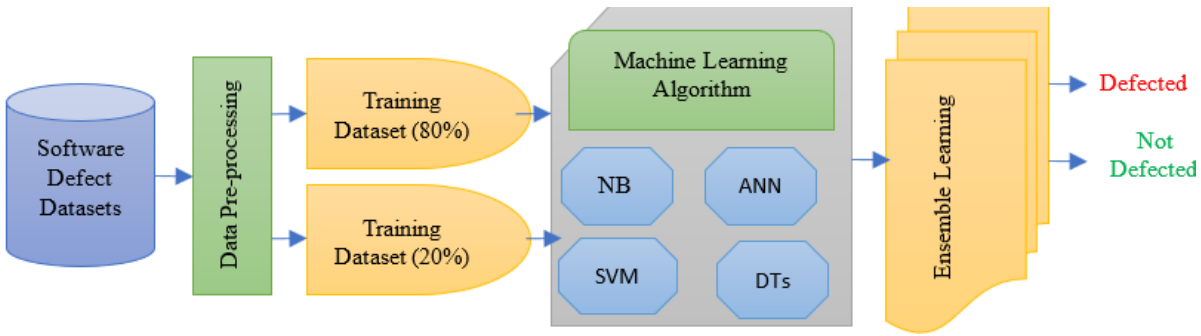

Figure 1. Proposed research methodology

The proposed research methodology focuses on developing a hybrid learning approach for software defect prediction. The methodology begins with data preprocessing, where raw datasets from the PROMISE repository undergo cleaning, normalization, and feature selection to enhance model performance. Selected features are then used to train multiple machines learning classifiers, including Decision Trees, Naïve Bayes, Artificial Neural Networks, and Support Vector Machines. These individual predictions are integrated using a hybrid learning technique, which employs methods like majority voting, weighted averaging, or stacking to improve classification accuracy. Finally, the hybrid model is evaluated using performance metrics such as precision, recall, accuracy, and F1-score, ensuring its effectiveness in detecting software defects.

Step 1: Dataset Splitting: The dataset DDD is divided into a training set for model learning and a test set for evaluation. This ensures that models generalize well to new data and do not overfit. The split is typically 70-80% for training and 20-30% for testing, using methods like random sampling or stratified sampling for class balance.

Step 2: Feature Selection: To enhance model performance, feature selection identifies the most relevant attributes while removing redundant or noisy features. This can be done using filter methods (statistical ranking), wrapper methods (iterative model-based selection), or embedded methods (built-in feature importance).

Step 3: Model Training and Prediction: Multiple classifiers (C1,C2,...,Ck) are trained on $Train_{new}$ and $Test_{new}$. Each classifier produces independent predictions, which are stored in a list Pred for later integration.

Step 4: Hybrid Integration: To improve fault prediction accuracy, the individual model predictions in Pred are combined using majority voting, weighted averaging, or stacking to form the hybrid model C_{hybrid}. This hybrid approach leverages the strengths of multiple classifiers to enhance overall reliability.

Step 5: Model Evaluation: The hybrid model $C_{hybrid}$is assessed on D_{test} using performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. This evaluation ensures that the hybrid model performs well before being deployed for real-world predictions.

Step 6: Final Prediction: The trained hybrid model$C_{hybrid}$is used to predict fault labels on unseen test data, generating. Final_Predictions.This step ensures the model is capable of detecting faults in new scenarios.

Step 7: Output Results: Finally, the hybrid fault prediction model C_{hybrid} and its predicted fault labels are returned as output. These predictions assist in proactive fault detection, improving system reliability and maintenance efficiency.

5. DATASET

The study utilizes datasets from NASA, which have gained significant attention in recent years for software engineering research. These datasets, outlined in the accompanying table, are essential for analyzing various software fault prediction aspects. Pre-processing is a crucial step to ensure compatibility with classification algorithms, involving normalization of numerical values and imputation of missing data. The experiments use datasets from the PROMISE repository, which contain real NASA software projects and diverse software modules. Benchmarking with these publicly available datasets enables comparisons with existing research efforts. The datasets include various code metrics, such as McCabe's cyclomatic complexity, code length, and Halstead's complexity, which help assess software quality. The NASA MDP datasets use a binary target variable, where '1' indicates a defect and '0' signifies no defect.

This study employs three datasets—Dataset1, Dataset2, and Dataset3—each containing key measurements for evaluating software fault prediction models. All datasets include two primary variables: the number of faults (Fi) and the number of test workers (Ti) recorded daily (Di) during different software testing phases. Dataset1, Dataset2, and Dataset3 contain 50, 100, and 150 measurements, respectively, offering insights into fault occurrences during the testing phase. Notably, Dataset3 is based on real-world data from a test/debug program for a real-time control application, providing practical insights into debugging-related faults.

To enhance analysis, the datasets underwent preprocessing using a novel clustering technique, which assigns class labels to fault data. These faults are categorized into five classes (A, B, C, D, and E), as detailed in Table 1, which presents class values and instance distributions. The study evaluates machine learning algorithms for software bug prediction using confusion matrix-based performance metrics. The following sections will provide a detailed discussion on the confusion matrix and specific evaluation metrics used in this research.

Table 1: Distribution of Fault Data Across Different Fault Classes

Fault Class	Number of Faults	DS1	DS2	DS3
A	0-5	18	52	87
B	6-11	22	28	35
C	12-17	5	10	18
D	18-23	3	6	5
E	More than 23	2	4	5
		50	100	150

6. PERFORMANCE EVALUATION PARAMETERS

The following sub-sections give the basic definitions of the performance parameters used for fault prediction (Table 2).

Table 2: Confusion Matrix for Classifying Modules as Faulty or Not Faulty

	Not-Faulty	Yes Faulty
Not-Faulty	Yes Negative (YN)	Not Positive (NP)
Yes Faulty	Not Negative (NN)	Yes Positive (YP)

The confusion matrix is categories into four categories:

1. Yes positives (YP) are the number of modules correctly classified as faulty modules.
2. Not positives (NP) refer to not-faulty classes incorrectly labeled as faulty classes.
3. Yes negatives (YN) correspond to not-faulty modules correctly classified as such.
4. Finally, Not negatives (NN) refer to faulty classes incorrectly classified as not-faulty classes.

According to authors in [1], following are the performance parameter used to measures the classification techniques.

Precision: It is used to measure the degree to which the repeated measurements under unchanged conditions show the same results.

$$Precision = \frac{TP}{FP+TP} \quad \text{.....(1)}$$

Recall: It indicates the how many of the relevant things which are to be identified. It is represented as:

$$Recall = \frac{TP}{FN+TP} \text{.....(2)}$$

F-Measure: These are used to join the precision and recall numeric value to produce one score, which is defined as the harmonic mean of the recall and precision. It is computed as:

$$F - Measure = \frac{2*Recall*Precision}{Recall+Precision} \text{... .(3)}$$

Specificity: Specificity indicates how effectively a classifier identifies the negative labels. It may be expressed as:

$$Specificity = \frac{TN}{FP+TN} \text{.....(4)}$$

Accuracy: Accuracy measure is the proportion of predicted fault prone modules that are inspected out of all modules. It is defined as:

$$Accuracy = \frac{TN+TP}{TP+TN+FP+FN} \text{.....(5)}$$

7. RESULT AND ANALYSIS

The JM1 dataset is a widely used benchmark in software defect prediction and is part of NASA's Metrics Data Program (MDP). It contains software metrics collected from a large-scale NASA project, where each instance represents a software module described by attributes such as lines of code, cyclomatic complexity, and other software engineering metrics. The primary objective of this dataset is to predict whether a module contains defects, making it valuable for evaluating machine learning models in defect detection. Given its real-world origin, the JM1 dataset presents a challenging and realistic environment for testing predictive algorithms. Similarly, the CM1 dataset, also from NASA's MDP, is derived from a different project but includes comparable software metrics, such as code complexity measures. CM1 is often analyzed alongside JM1 to compare the generalizability of machine learning models across different projects. Each instance represents a software module, with the goal of identifying defects, making CM1 an essential resource for testing the robustness of predictive models.

The PC1 dataset is another component of NASA's MDP, containing software metrics from yet another NASA project. Like JM1 and CM1, PC1 is used for software defect prediction at the module level and includes a variety of software engineering attributes, such as complexity, code churn, and coupling metrics. Unlike JM1 and CM1, PC1 is relatively smaller in size, which presents unique challenges for machine learning models, particularly in terms of avoiding overfitting. Despite its smaller scale, PC1 is frequently used with other MDP datasets to assess the transferability and effectiveness of defect prediction models across different software projects. Collectively, these datasets provide a comprehensive testbed for evaluating machine learning algorithms in software defect prediction, allowing researchers to analyze how well models generalize across different projects and software metrics.To perform statistical analysis, bugs were collected from Promise data repository [9]. Table 3 shows the distribution of bugs based on the number of occurrence (in terms of percentage of class containing number of bugs.

Table 3: Distribution of Bugs in PROMISE Dataset [9]

No. of Classes	% of Bugs	NumberofAssociated Bugs
791	79.33	0
138	13.84	1
31	3.1	2
15	1.5	3
8	0.8	4
2	0.2	5
4	0.4	6
3	0.3	7
3	0.3	8
2	0.2	9

Proposed case study shown in Figure 2 contains 997 number of different classes in which 79.33% of classes contain zero bugs i.e., out of 997 classes: 791 classes contains zero bugs, 13.84% of classes contain at least one bug, 3.10% of classes contain a minimum of two bugs, 1.50% of classes contain three bugs, 0.80% classes contain four bugs, 0.20% of classes contain five and nine bugs, 0.40% classes contain six bugs, 0.30% of classes contain seven and eight bugs.

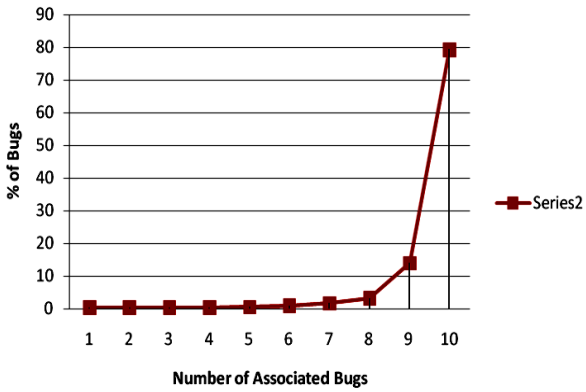

Figure 2: Distribution of bugs in %age

The performance analysis of various machine learning algorithms across the JM1, CM1, and PC1 datasets highlights a clear hierarchy based on precision, recall, accuracy, and F1-score. Among all models, Hybrid Learning (EL) consistently achieves the highest scores, ranging between 0.96 and 0.99, demonstrating its ability to effectively balance precision and recall, leading to superior overall accuracy and robustness. Following EL, the Decision Tree (DT) algorithm also performs well, with scores ranging from 0.94 to 0.97. While DT is highly effective and particularly useful for interpretability, it is slightly less precise than EL.

The Naïve Bayes (NB) algorithm, known for its simplicity and efficiency, delivers moderate performance, with scores between 0.92 and 0.96. While it lags slightly behind DT, NB remains a viable option due to its balance between computational efficiency and accuracy. Artificial Neural Networks (ANNs), despite their power in many applications, show slightly lower performance in this study, with scores between 0.91 and 0.95, suggesting that ANNs may require fine-tuning or larger datasets for optimal performance. Lastly, Support Vector Machines (SVMs) exhibit the lowest performance, with scores ranging from 0.90 to 0.93. While SVMs are often effective in high-dimensional spaces, they

appear less capable of handling the complexity of these datasets. In conclusion, Hybrid Learning (EL) emerges as the most effective model, followed by Decision Trees, Naïve Bayes, Artificial Neural Networks, and Support Vector Machines, in descending order of performance.

The performance analysis of machine learning algorithms across the JM1, CM1, and PC1 datasets reveals that Hybrid Learning (HL) consistently outperforms all other models. HL achieves the highest precision, recall, accuracy, and F1-score across all datasets, with precision ranging from 0.97 to 0.99 and recall between 0.96 and 0.98. Its superior accuracy (0.97 to 0.99) indicates a strong ability to correctly classify defective and non-defective modules, making it the most reliable model for software defect prediction. Following HL, the Decision Tree (DT) algorithm also demonstrates strong performance, with accuracy scores between 0.95 and 0.97. While slightly lower than HL, DT remains effective, especially in scenarios where interpretability and rule-based decision-making are essential.

The Naïve Bayes (NB) algorithm performs moderately well, with precision ranging from 0.93 to 0.96 and accuracy between 0.94 and 0.96. Although slightly behind DT, NB maintains a good balance between computational efficiency and classification accuracy. Artificial Neural Networks (ANNs) show similar trends, achieving accuracy between 0.93 and 0.95, though they may require additional fine-tuning for optimal performance. Finally, Support Vector Machines (SVMs) exhibit the lowest performance, with accuracy scores between 0.91 and 0.93, indicating that while SVMs can be effective in certain contexts, they struggle to handle the complexity of these datasets. Overall, the hierarchy of performance ranks HL as the best model, followed by DT, NB, ANN, and SVM in decreasing order of effectiveness.

Table 4: Performance Evaluation of the Proposed Hybrid Learning Algorithm

Algorithm	Performance Measurement	Dataset		
		JM1	CM1	PC1
Hybrid Learning (HL)	Precision	0.99	0.98	0.97
	Recall	0.98	0.97	0.96
	Accuracy	0.99	0.98	0.97
	F1	0.98	0.97	0.96
Decision Tree (DT)	Precision	0.97	0.96	0.95
	Recall	0.96	0.95	0.94
	Accuracy	0.97	0.96	0.95
	F1	0.96	0.95	0.94
Naive Bayes (NB)	Precision	0.96	0.95	0.93
	Recall	0.95	0.94	0.92
	Accuracy	0.96	0.95	0.94
	F1	0.95	0.94	0.93
Artificial Neural Network (ANN)	Precision	0.95	0.94	0.92
	Recall	0.94	0.93	0.91
	Accuracy	0.95	0.94	0.93
	F1	0.94	0.93	0.92
Support Vector Machine (SVM)	Precision	0.93	0.92	0.91
	Recall	0.92	0.91	0.90
	Accuracy	0.93	0.92	0.91
	F1	0.92	0.91	0.90

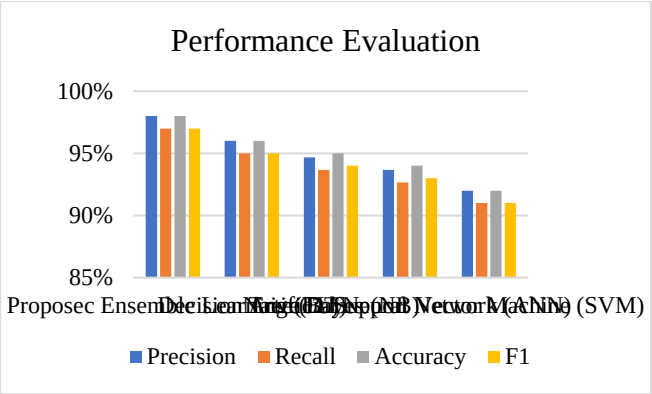

Figure 3: Performance Evaluation of the Proposed Hybrid Learning Algorithm

8. CONCLUSION

The findings of this study highlight the effectiveness of various machine learning algorithms in software defect prediction using the JM1, CM1, and PC1 datasets. Among the evaluated models, Hybrid Learning (HL) consistently achieved the highest precision, recall, accuracy, and F1-score, demonstrating its robustness in identifying software defects with minimal misclassification. The Decision Tree (DT) algorithm also performed well, offering high accuracy and interpretability, making it a strong alternative to HL. Naïve Bayes (NB) and Artificial Neural Networks (ANNs) provided moderate performance, showing potential for defect prediction but requiring optimization to match the efficiency of HL and DT. On the other hand, Support Vector Machines (SVMs) exhibited the lowest performance, suggesting that they may not be well-suited for handling the complexity of the given datasets.

Overall, this study underscores the importance of selecting an appropriate machine learning model based on dataset characteristics and predictive performance. The superior results of HL suggest that hybrid approaches, which integrate multiple learning paradigms, can enhance defect prediction accuracy and reliability. These findings contribute to ongoing research in software quality assurance, offering insights into how machine learning can be leveraged for more effective defect detection. Future work could explore further optimization of hybrid models, integration with additional datasets, and real-world validation to improve generalizability across different software development environments.

REFERENCES

1. S. Montani and C. Anglano, “Achieving self-healing in service delivery software systems by means of case-based reasoning,” Appl. Intell., vol. 28, no. 2, pp. 139–152, Apr. 2008.
2. Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, “An Empirical Study on the Effectiveness of Feature Selection for Cross-Project Defect Prediction,” IEEE Access, vol. 7, pp. 35710–35718, 2019.
3. J. Hotzkow and Jenny, “Automatically inferring and enforcing user expectations,” in Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2017, 2017, pp. 420–423.
4. A. A. Hudaib, H. N. Fakhouri, A. A. Hudaib, and H. N. Fakhouri, “An Automated Approach for Software Fault Detection and Recovery,” Commun. Netw., vol. 08, no. 03, pp. 158–169, Jul. 2016.
5. J. Moeyersoms, E. Junqué de Fortuny, K. Dejaeger, B. Baesens, and D. Martens, “Comprehensible software fault and effort

prediction: A data mining approach," J. Syst. Softw., vol. 100, pp. 80–90, Feb. 2015.
6. Z. Xu et al., "Software defect prediction based on kernel PCA and weighted extreme learning machine," Inf. Softw. Technol., vol. 106, pp. 182–200, Feb. 2019.
7. R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," Appl. Soft Comput., vol. 21, pp. 286–297, Aug. 2014.
8. Thota, M.K., Shajin, F.H. and Rajesh, P. (2020) Survey on Software Defect Prediction Techniques. International Journal of Applied Science and Engineering, 17, 331-344.
9. Li, N., Shepperd, M. and Guo, Y. (2020) A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction. Information and Software Technology, 122.
10. Matloob, F., Ghazal, T.M., Taleb, N., Aftab, S., Ahmad, M., Khan, M.A. and Soo mro, T.R. (2021) Software Defect Prediction Using Hybrid Learning: A Systematic Literature Review. IEEE Access, 9, 98754-98771.
11. Akimova, E.N., Bersenev, A.Y., Deikov, A.A., Kobylkin, K.S., Konygin, A.V., Me zentsev, I.P. and Misilov, V.E. (2021) A Survey on Software Defect Prediction Using Deep Learning. Mathematics, 9, Article No. 1180.
12. Khan, M.A., Elmitwally, N.S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M. and Khan, F. (2022) Software Defect Prediction Using Artificial Neural Networks: A Systematic Literature Review. Scientific Programming, 2022,
13. Goyal, S. (2022) Effective Software Defect Prediction Using Support Vector Ma chines (SVMs). International Journal of System Assurance Engineering and Man agement, 13, 681-696.
14. Stradowski, S. and Madeyski, L. (2023) Machine Learning in Software Defect Pre diction: A Business-Driven Systematic Mapping Study. Information and Software Technology, 155,
15. V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, "EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES," Int. J. Artif. Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.
16. Hernández-Molinos, M.J., Sánchez-García, A.J., Barrientos-Martínez, R.E., Pérez- Arriaga, J.C. and Ocharán-Hernández, J.O. (2023) Software Defect Prediction with Bayesian Approaches. Mathematics, 11.

