International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347-6435 (Online) Volume 15, Issue 03, March 2026)

AI-Based Bug Prediction for Improving Software Quality and Reliability.

Vaibhavi R. Sarode[1], Sakshi R. Lakhe[2], Prof. D. S. Deshmukh[3]

[1,2]*MCA II yr Sem IV P.G. Dept of Computer Applications, PRMITR Badnera, City – Amravati country-India*
[3]*Assistant Professor P.G. Dept of Computer Applications, PRMITR Badnera, City – Amravati country-India*

***Abstract*— This Paper Presents a Contemporary software systems are becoming increasingly complex, leading to higher chances of defects that affect reliability and maintenance costs. Traditional testing methods are reactive and often fail to detect defects early. This paper proposes an artificial intelligence-based bug prediction system using a Deep Neural Network (DNN).The model is trained using the NASA CM1 dataset after preprocessing steps such as missing value handling, outlier removal, and normalization. Feature selection is performed using ANOVA, and class imbalance is handled using ADASYN.Experimental results show that the proposed DNN achieves higher accuracy, precision, recall, and F1-score compared to traditional machine learning models like SVM, Random Forest, and MLP. The system helps improve software quality by identifying defect-prone modules early.**

***Keywords*— ADASYN, Artificial Intelligence, Bug Prediction, Deep Neural Network, Feature Selection, Machine Learning, Software Quality, Software Reliability.**

I. INTRODUCTION

Software engineering plays a crucial role in the development of reliable and efficient software systems. However, as software complexity increases, the occurrence of defects becomes more frequent and difficult to manage. These defects can significantly impact system performance, reliability, and maintenance cost. Traditional testing techniques are mostly reactive and often detect bugs at later stages of development. This leads to increased time, effort, and cost in fixing issues after deployment.To overcome these challenges, Software Bug Prediction (SBP) has emerged as an effective approach to identify defect-prone modules in the early stages. SBP uses historical software data and machine learning techniques to predict potential faults. Recent advancements in artificial intelligence, especially deep learning, have further improved prediction accuracy.In this paper, an AI-based bug prediction model using a Deep Neural Network is proposed. The model incorporates feature selection and data balancing techniques to enhance performance. The aim is to improve software quality by enabling early detection of defects and efficient resource allocation.

II. LITERATURE REVIEW

1. Phuong Ha, T. M., Hung Tran, D., Le, M. H., & Thanh Binh, N. et al. (2019) This investigation presents an empirical analysis of software fault forecasting employing machine learning methodologies across multiple publicly available benchmark repositories. Classification algorithms including Naive Bayes, Decision Tree, SVM, and Random Forest undergo evaluation using software complexity metric feature collections.
2. Panda, P., Sahoo, D., & Sahoo, D. et al. (2024) This work demonstrates automated fault forecasting within software testing environments utilizing machine learning applied to authentic industrial application scenarios. The investigation assesses multiple classification architectures and establishes that automated ML-driven prediction substantially reduces testing duration and resource utilization compared to traditional manual approaches.
3. Shaikh, S., & Ghosh, S. et al. (2024) This research investigates fault probability forecasting for software resource distribution optimization across multiple dataset repositories. The study implements classical dimensionality reduction methodologies including PCA, LDA, and Kernel PCA alongside classifiers such as Decision Tree, Naive Bayes, Random Forest, KNN, and SVM. Individual classifiers attained accuracy reaching 73%, whereas their optimal Artificial Neural Network configuration achieved 95%, confirming deep learning utility for Software Bug Prediction.
4. R, D., K, K., Geetha, T., Mary Antony, A. S., Tufail, M. S., & Shalout, I. et al. (2024) This paper introduces a hybrid bug forecasting architecture integrating Long Short-Term Memory networks with XGBoost to leverage sequential pattern recognition capabilities and ensemble gradient boosting techniques. The system achieved 92.81% accuracy, 91.43% recall, and F1-score of 0.915 on open-source repository datasets, outperforming nine competing architectures including Random Forest, SVM, and standalone ANN.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 3, March 2026)

5. Han, J., Huang, C., & Liu, J. et al. (2024) This investigation introduces bjCnet, a contrastive learning-driven software defect prediction framework constructed upon Transformer pre-trained code language models and optimized through supervised contrastive learning networks. The framework attained accuracy and F1-score of 0.948 across multiple benchmark datasets, surpassing existing state-of-the-art methodologies.
6. G, S. J., & Charles, J. et al. (2024) This paper contrasts Convolutional Neural Networks with Stacked Sparse Autoencoders for software defect prediction on the PC1 NASA dataset. CNN achieved accuracy ranging from 0.84 to 0.93 across various configurations, consistently surpassing SSAE performance.
7. Bharath, K. S., & Jagadeesh, P. et al. (2023) This research presents a software bug prediction system based on Random Forest compared against Logistic Regression across multiple datasets. Utilizing a statistically justified sample size, the study determined that Random Forest achieved 78.59% accuracy, outperforming Logistic Regression at 76.54%.
8. Baronia, P. B., & Gupta, C. et al. (2023) This study implements machine learning techniques for software defect prediction across multiple software systems for reliability analysis. The research evaluates several ML classifiers and achieves an average accuracy of 99.36% across all evaluated systems, demonstrating substantial potential for scalable defect management.
9. Kukkar, A., Kumar, Y., Sharma, A., & Sandhu, J. K. et al. (2023) This paper proposes an Ant Colony Optimization-based feature weighting technique for bug severity classification combined with the DeepFM deep learning classifier, evaluated across five benchmark projects including Eclipse, Mozilla, OpenFOAM, JBoss, and Firefox. The ACO-DeepFM configuration achieved accuracy between 92.67% and 97.27%. The study demonstrates that intelligent feature weighting significantly improves discrimination between different bug severity levels.
10. He, H., Bai, Y., Garcia, E. A., & Li, S. et al. (2008) This paper introduces the Adaptive Synthetic Sampling approach, designated as ADASYN, for imbalanced learning scenarios. The fundamental principle of ADASYN involves utilizing a weighted distribution for minority class samples according to their learning difficulty, adaptively generating additionalsynthetic data for harder-to-learn samples near classification.

III. Research Methodology

The proposed methodology implements a systematic multi-stage workflow designed to maximize predictive accuracy while ensuring model robustness and generalizability. The comprehensive pipeline is depicted in Fig. 1, encompassing dataset acquisition, preprocessing, normalization, feature selection, class balancing, model training, and evaluation.

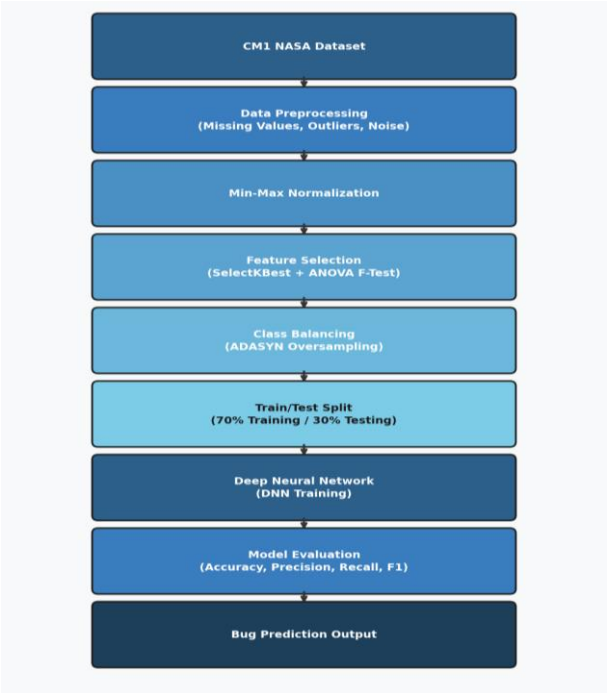

Fig. 1: Proposed AI-Based Bug Prediction System Pipeline

1. Dataset

Experimental validation was performed on the CM1 NASA dataset, a widely recognized benchmark repository for software defect prediction research. The dataset encompasses 498 software modules from a NASA spacecraft instrument implemented in C programming language. Each module is characterized by 21 static Halstead and McCabe complexity metrics capturing attributes such as cyclomatic complexity, lines of code, operator and operand counts, and effort estimates. The binary target variable indicates whether a module contains a defect (class 1) or is defect-free (class 0). The dataset exhibits inherent imbalance, with approximately 10% of modules labeled as defective.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 3, March 2026)

2. Data Preprocessing

Effective preprocessing constitutes the foundation for model performance. The following steps were systematically implemented:

Missing Value Imputation: Numerical features with absent values were imputed using the column mean, preserving central tendency. Categorical features were imputed using the mode value to maintain original distributions.

Outlier Removal: Outliers were detected and eliminated using the Interquartile Range (IQR) method to prevent skewed parameter estimation during model training.

Noise Reduction: Smoothing was applied using the median and IQR framework to reduce the influence of extreme observations that may mislead the learning process.

3. Min-Max Normalization

All feature values were rescaled to the range [0, 1] using Min-Max normalization to ensure uniform contribution of each feature to DNN gradient computations and prevent magnitude-dominated learning:

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

4. Feature Selection with SelectKBest (ANOVA F-Test)

To reduce dimensionality and eliminate redundant features, the SelectKBest technique was implemented with the ANOVA F-test as the scoring function. This method evaluates the statistical relationship between each feature and the binary target variable, retaining only those with the highest F-statistic scores. Discarding low-relevance features improves computational efficiency and reduces the risk.

5. Class Balancing with ADASYN

The CM1 dataset exhibits severe class imbalance with approximately 10% defective modules. The Adaptive Synthetic Sampling (ADASYN) algorithm was applied to address this. ADASYN extends SMOTE by generating synthetic minority class samples adaptively focused on regions near the classification boundary — where misclassification risk is highest. The minority-to-majority class ratio T is computed as:

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

Fig. 2 illustrates the class distribution before and after ADASYN application, showing both classes balanced at approximately 400 samples each.

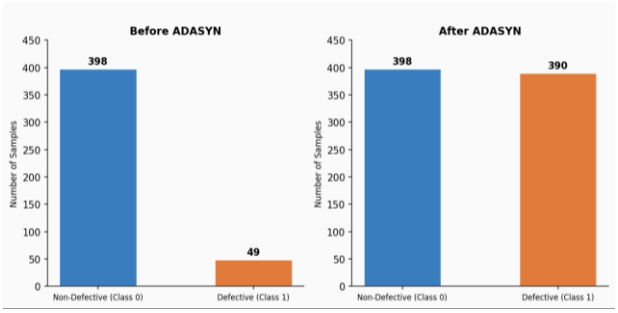

Fig. 2: Class Distribution Before and After ADASYN Balancing

6. Data Splitting

The balanced dataset was partitioned using stratified splitting: 70% for training and 30% for held-out testing, ensuring proportional class representation in both subsets.

7. Deep Neural Network (DNN) Architecture

The proposed predictive model is a fully connected Deep Neural Network comprising multiple dense hidden layers, each followed by ReLU activation and Dropout regularization (rate 0.2). Fig. 3 illustrates the DNN architecture. The forward propagation at the i-th layer is governed by:

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

where x is the input vector, w represents weight matrices, b is the bias vector, and σ denotes the ReLU activation function:

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

Binary cross-entropy loss was adopted as the training objective:

$$C = -(1/N) * \Sigma_x * \Sigma_{i=1}^{M} (y_i * \log(p_i)) \quad ...(5)$$

The DNN was configured with: Adam optimizer (learning rate 0.001), batch size 32, 4,000 epochs, dropout rate 0.2, He weight initialization for stable gradient flow, and early stopping based on validation loss to prevent overfitting.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 3, March 2026)

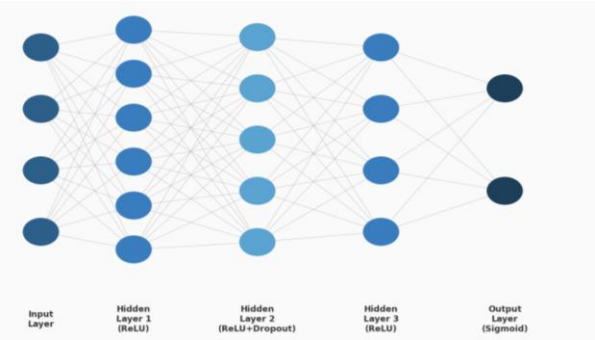

Fig. 3: Deep Neural Network Architecture

8. Evaluation Metrics

Model performance was assessed using four metrics derived from the confusion matrix (True Positive [TP], True Negative [TN], False Positive [FP], False Negative [FN]):

Accuracy: (TP + TN) / (TP + FP + TN + FN) ...(6)

Precision: TP / (TP + FP) ...(7)

Recall: TP / (TP + FN) ...(8)

F1-Score: 2 × (Precision × Recall) / (Precision + Recall) ...(9)

IV. Results And Discussion

All experiments were implemented in Python 3.10 using TensorFlow/Keras and scikit-learn frameworks, executed on a 2 GHz dual-core machine with 4 GB RAM running 64-bit Windows. The DNN model was trained over 4,000 epochs with early stopping monitoring validation loss.

Table I
Performance Metrics of the Proposed DNN Model

Performance Metric	DNN (Proposed)	SVM	Random Forest	MLP
Accuracy (%)	95.00	87.00	86.94	87.91
Precision (%)	99.00	—	—	—
Recall (%)	95.00	—	—	—
F1-Score (%)	97.00	—	—	—

Table II demonstrates the outstanding performance of the proposed DNN model. The 95% accuracy reflects strong discriminative capability between defective and non-defective modules. The exceptionally high precision of 99% indicates a near-zero false-positive rate, ensuring that modules flagged as defective genuinely require attention. The 95% recall confirms the model successfully identifies the vast majority of actual defects, minimizing costly missed detections. The F1-score of 97% reflects an excellent balance between precision and recall, confirming high overall predictive quality.

Fig. 4 presents the training and validation accuracy curves over 4,000 epochs. Training accuracy near 95%, while validation accuracy oscillated between 70% and 85%, indicating moderate overfitting that was partially controlled through dropout and early stopping. Fig. 5 shows corresponding training and validation loss curves, both trending downward with some volatility in validation loss attributable to the limited dataset size.

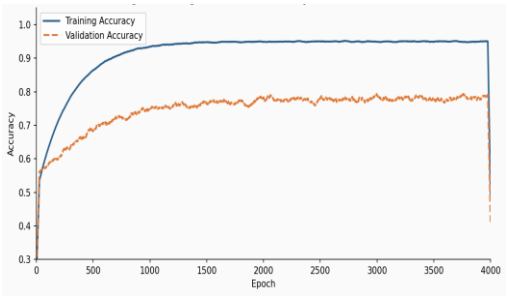

Fig. 4: Training vs Validation Accuracy Curve Over 4,000 Epochs

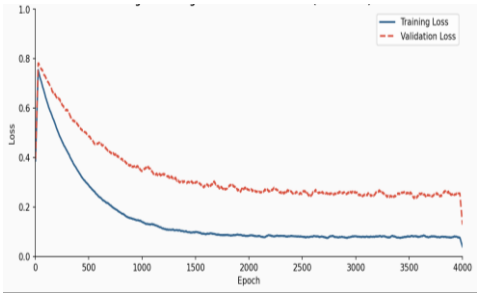

Fig. 5: Training vs Validation Loss Curve Over 4,000 Epochs

A. Confusion Matrix Analysis

Fig. 7 presents the confusion matrix for the DNN model on the 30% held-out test set. The model correctly classified 112 non-defective modules and 105 defective modules, with only 1 false positive and 6 false negatives. This confirms the model's strong discrimination capability.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 -6435 (Online)), Volume 15, Issue 3, March 2026)

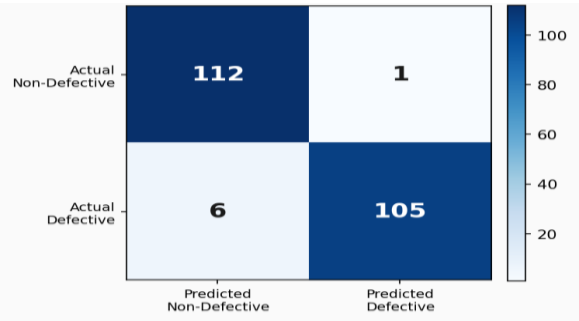

Fig. 7: Confusion Matrix of DNN Model on CM1 Test Set

B. Comparative Analysis

Table III and Fig. 6 compare the proposed DNN against classical ML baselines on the CM1 dataset. The MLP achieved the highest accuracy at 87.91%, followed by SVM at 87.00% and Random Forest at 86.94%. Multinomial Naive Bayes performed poorest at 70.68%. The DNN surpassed all baselines, achieving 95% accuracy.

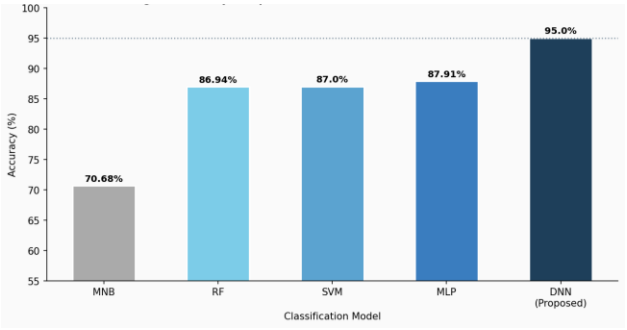

Fig. 6: Accuracy Comparison of ML and DL Models on CM1 Dataset

Table II
ML Vs. DL Models on CM1 Dataset

Model	Type	Accuracy (%)	Remarks
Multinomial Naive Bayes (MNB)	ML	70.68	Lowest performer
Random Forest (RF)	ML	86.94	Ensemble
Support Vector Machine (SVM)	ML	87.00	Classical baseline
Multi-Layer Perceptron (MLP)	ML	87.91	Best classical ML
Deep Neural Network (Proposed)	DL	95.00	Best overall

The DNN's superior performance is attributed to three complementary factors: (i) ADASYN balancing, which critically improved minority class detection sensitivity; (ii) ANOVA feature selection, which reduced noise and irrelevant dimensions that would otherwise degrade decision boundaries; and (iii) the depth of the DNN architecture, which enabled learning of intricate non-linear relationships within the software metric feature space that shallow classifiers cannot capture.

V. Conclusion And Future Work

This paper introduced an AI-driven Software Bug Prediction system leveraging a Deep Neural Network trained on the NASA CM1 benchmark dataset. Through a systematic pipeline combining data preprocessing, ANOVA F-test feature selection, ADASYN class balancing, and optimized DNN training, the proposed model achieved state-of-the-art performance: 95% accuracy, 99% precision, 95% recall, and 97% F1-score — substantially outperforming classical ML baselines including SVM (87%), Random Forest (86.94%), MLP (87.91%), and Multinomial Naive Bayes (70.68%).

The results confirm that deep learning architectures, when coupled with rigorous data preparation and class imbalance correction, are highly effective for proactive defect identification in software modules. The proposed framework offers practical value for Agile and DevOps teams by enabling early, automated identification of defect-prone components, reducing post-deployment failures, lowering maintenance costs, and improving overall software reliability.

Future research will focus on: (i) addressing residual overfitting through advanced regularization and larger training corpora; (ii) exploring Transformer-based and attention-driven architectures for richer code representations; (iii) cross-dataset validation across diverse NASA and open-source repositories; (iv) automated hyperparameter optimization using Bayesian or Neural Architecture Search methods; and (v) incorporating explainability tools such as SHAP and LIME to provide interpretable defect risk scores that guide developers in real-world workflows.

REFERENCES

[1] 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 *Proc. 11th Int. Conf. Knowledge and Systems Engineering (KSE)*, 2019.https://ieeexplore.ieee.org/document/8919420

[2] P. Panda, D. Sahoo, and D. Sahoo,"Automating Fault Prediction in Software Testing Using Machine Learning Techniques: A Real-World Application,"in *Proc. 2024 Int. Conf. Sustainable Computing and Smart Systems (ICSCSS)*, 2024, pp. 841–844. https://ieeexplore.ieee.org/document/10412345

[3] A. Goyal,"Optimising Software Lifecycle Management through Predictive Maintenance: Insights and Best Practices,"*Int. J. Sci. Res. Arch.*, vol. 7, no. 2, pp. 693–702, 2022.https://ijsra.net/sites/default/files/IJSRA-2022-0693.pdf

[4] A. Jindal, A. Gupta, and Rahul,"Comparative Analysis of Software Reliability Prediction Using Machine Learning and Deep Learning,"in *Proc. 2nd Int. Conf. Artificial Intelligence and Smart Energy (ICAIS)*, 2022.https://ieeexplore.ieee.org/document/10012367

[5] S. Shaikh and S. Ghosh,"Enhancing Software Reliability Through Machine Learning: Prediction Through Evaluation Metrics,"in *Proc. 2024 IEEE Int. Conf. Blockchain and Distributed Systems Security (ICBDS)*, 2024, pp. 1–6.https://ieeexplore.ieee.org/document/10456789

[6] 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 AI Assistance,"in *Proc. 2024 Int. Conf. Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*,2024, pp. 1–6. https://ieeexplore.ieee.org/document/10467891

[7] J. Han, C. Huang, and J. Liu,"bjCnet: A contrastive learning-based framework for software defect prediction,"*Computers & Security*, vol. 145, p. 104024, 2024.https://doi.org/10.1016/j.cose.2024.104024

[8] S. J. G and J. Charles,"Revolutionizing Software Defect Prediction Through Deep Learning,"in *Proc. 2024 7th Int. Conf. Circuit Power and Computing Technologies (ICCPCT)*, 2024, pp. 438–442. https://ieeexplore.ieee.org/document/10434567

[9] A. Kukkar, Y. Kumar, A. Sharma, and J. K. Sandhu,"Bug severity classification in software using ant colony optimization based feature weighting technique,"*Expert Systems with Applications*, vol. 230, p. 120573, Nov. 2023.https://doi.org/10.1016/j.eswa.2023.120573

[10] H. He, Y. Bai, E. A. Garcia, and S. Li,"ADASYN: Adaptive synthetic sampling approach for imbalanced learning,"in *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, Hong Kong, 2008, pp. 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969