

Use of deep learning in early software bug detection

Syed Mibran Hassan Zaidi ^a, Maria Khan ^b, Dr. Mustafa Latif ^c, Ali Akhtar ^d, Sallar Khan ^e

^a Master's Researcher of Computer Science & Information Technology Department, NED University of Engineering & Technology, Karachi 75270, Pakistan

^b Master's Researcher of Computer Science & Information Technology Department, NED University of Engineering & Technology, Karachi 75270, Pakistan

^c Associate Professor in the Department of Software Engineering, NED University of Engineering & Technology, Karachi 75270, Pakistan

^d Lecturer in the Department of Computer Science & Information Technology, NED University of Engineering & Technology, Karachi 75270, Pakistan

^e Doctoral Researcher School of Informatics and Cybersecurity, Technological University Dublin, Dublin D15 YV78, Ireland

* Corresponding Author: Syed Mibran Hassan Zaidi, Email: mibranhassan@gmail.com

Received: 25 April 2025, Revised: 23 June 2025, Accepted: 30 June 2025, Published: 01 July 2025

KEYWORDS	ABSTRACT
Deep Learning CNN Random Forest Software Bug Detection Ensemble Approach Voting Ensemble	In the software maintenance and development process, software bug identification is a significant challenge because this is connected with the successful software in its whole state. To increase software efficiency, dependability, and quality early software problem identification is important. Therefore, an efficient software bug detection model is a critically significant challenging operation, and that has been developed in this work by designing the efficient model while using the promises dataset as input for bug identification. An ensemble technique approach is proposed to increase the performance of detecting software bugs for moderate to large datasets. This was achieved by using an ensemble parallel technique for training the Convolutional Neural Network and Random Forest algorithm in a parallel way. Then, the predictions from both models are aggregated using an ensemble voting technique. The performance of the implied technique is evaluated using various kinds of metrics, including accuracy, F-measure, recall, and precision. Experimental findings show significant improvements in prediction accuracy and F1 scores compared to the standalone CNN model, including increases of 33.06% for Camel, 13.43% for Jedit, 50.85% for Xerces, and 18.35% for Synapse. These findings highlight the potential of ensemble learning techniques for enhancing software bug detection.

1. Introduction

Software maintenance is an essential component of software development. Bug identification is a key component of the maintenance cycle. The high quantity of bugs in software has made bug resolution challenging. Over time, important software configurations can lead to increased annoyance and decreased efficiency, impacting performance. A software bug refers to a mistake or fault that causes

unpredictable or inaccurate consequences. The occurrence of this issue has a direct impact on the framework's quality.

Software defects can occur when the final result does not fulfil the client's requirements or assumptions. SDP[25-27] is a technique that builds a classifier and predicts the code areas that potentially contain defects. When it comes to guiding the software testing process, accurate software defect prediction

can be very helpful. Software flaws that are prone to errors are identified using error expectation techniques structures. Software faults have a substantial impact on quality, reliability, and maintenance costs. Early identification of a product fault allows software to be modified to changing settings, resulting in better utilization of resources. Testing and debugging in computer programming are quite expensive and demand a significant number of assets. Efficient defect prediction reduces test expenses and may enhance software quality. The choice of an ensemble approach further reflects the commitment to refining predictive accuracy, robustness, and reliability. By integrating the strengths of different models, the ensemble approach improves the software defect prediction model's overall performance. The motivation for employing such a comprehensive methodology is grounded in the pursuit of a sophisticated and reliable tool that can contribute significantly to the early identification of bug cases within diverse software projects.

Furthermore, the motivation lies in filling the gaps in current literature and technologies to make a real impact in the field of software. The successful deployment of a deep learning-based ensemble model can open up new doors to precision in the identification of software bugs.

In this research, a challenge was encountered in the efficient detection of software bugs and the accuracy issue with moderate to large datasets, the goal was to implement a deep learning algorithm for software bug detection which have higher accuracy. While deep learning algorithms are recognized for their high utility with superior results are generally achieved with larger datasets. Confronted with these limitations, this resulted in the development of a model that achieves high performance, thereby improving its applicability for software bug detection with high accuracy across moderate to large datasets.

The primary contributions of the stated model are as follows:

In addressing the challenge of Accuracy, and moderate to large datasets for software bug detection, this thesis makes a noteworthy contribution by proposing and implementing an ensemble parallel technique, by incorporating the Convolutional Neural Network using the random forest technique. The predictions from both models are aggregated using an ensemble voting mechanism and this approach outperforms with respect to accuracy and F1-measure, proving its dominance in bug identification.

Development of an ensemble model, that combines CNN and RF, taking advantage of their unique

strengths CNN for complex hierarchical feature extraction and RF for robust classification.

The experiment was conducted on 14 various Promise repo datasets, these datasets were consolidated into seven larger datasets. For example, the datasets that have multiple versions are merged to create a single, moderate-to-large dataset. Various metrics are used for calculating the proposed model performance. The result states that the suggested model outperforms an additional existing model. This work not only expands the use of deep learning methods for software issue identification but also provides a practical solution for scenarios where data scarcity is a common challenge in software development and bug identification.

The research seeks to address the following questions:

- RQ1: How can effective techniques be combined to address the challenges of moderate to large datasets in enhancing software bug detection performance?

- RQ2: Does the proposed ensemble model integrating CNN and RF outperform traditional approaches in terms of accuracy and F-measure for software bug detection?

2. Literature Review

In recent years, rapid advancements computer technology and the extensive implementation of software systems have significantly provided bug-related data. Using the datasets, researchers have developed several models regarding bug detection, prediction, and classification with promising results. These studies used different methodologies in identifying and analyzing software issues. Some of the notable earlier research efforts in this domain are as follows:

2.1 Machine Learning & Software Bug Detection

As software defect prediction is so important in improving software reliability and quality, it has drawn a lot of attention recently. Various approaches have been proposed for fault prediction's efficacy and accuracy.

Olorunshola et al. [1] tested twelve machine-learning classification techniques on the PROMISE dataset. Their extensive research revealed that the Random Forest algorithm excelled the majority of others, but the Bayes Net classifier excelled in terms of false positive rate. However, the study's focus on a single dataset raised concerns about result generalizability.

Sanusi et al. [2] developed a software defect identification system that employs machine learning

methods to detect software issues quickly. Their study used Random Forest, Decision Tree, and Artificial Neural Network (ANN) methods, with Random Forest outperforming others in accuracy percentage (83.40), precision (53.18), and F-score (52.04). While the study demonstrated robust methodology through feature selection and cross-validation, it lacked detailed information about the datasets used and did not address the impact of class imbalance on algorithm performance.

Lucija Sikic et al. [3] introduced a new convolutional graph neural network (GCNN) that leverages an abstract syntax tree (AST) of the original code for issue prediction, adapting to various project sizes. This model provides high accuracy in terms of AUC and F-score and has good prediction ability.

Alkaberi and Assiri et al. [4] focused on predicting the number of software faults using CNN and multilayer perceptron (MLP) models. Their approach, which included oversampling techniques and log transformation, demonstrated improved performance over traditional regression models. However, the study's reliance on Java datasets with specific metrics limited its external validity.

Predicting software defects during the SDLC helps identify modules that require intensive testing. ML techniques, both supervised and unsupervised, are commonly used for prediction, complementing semi-supervised and reinforcement learning (Long et al. [5]).

Classification, a supervised machine learning technique, is widely used in prediction models, including weather forecasting, disease prediction, sentiment analysis, and software fault identification (Li & Ortigas White [6]). A comparison of four classifiers for software defect prediction (SDP) was carried out using NASA datasets (Daoud et al. [7]).

Jude and Uddin et al. [8] introduced a hybrid machine learning algorithm combining multiple models through stacking. Their approach, which integrated Explainable AI (XAI) techniques for interpretability, showed improved performance across various metrics using NASA-MD datasets. The study acknowledged limitations in generalizability and recommended further research on computational requirements.

Ali et al. [9] developed the Voting Ensemble-Based Software Fault Detection Model (VESDP), which combines four different supervised machine learning classifiers. The VESDP model outperformed twenty cutting-edge approaches on seven historical defect datasets, demonstrating exceptional accuracy. However, the study's focus on NASA MDP datasets

and emphasis on accuracy metrics potentially limited its broader applicability.

Ensemble learning (EL) combines predictions from multiple weak classifiers to create a stronger classifier that outperforms standalone classifiers (Mehta & Patnaik [11]). EL provides homogeneous classifiers such as bagging, boosting, and rotational forest, as well as heterogeneous classifiers like voting and stacking (Wu & Wang [12]). Voting is a diverse ensemble classifier that brings together predictions from multiple basic classifiers.

Javed [13] suggested a software defect prediction framework that uses a nested EL approach with voting as the master classifier and three base ensemble classifiers (bagging, boosting, and stacking). The suggested system achieved 83.46% and 79.65% accuracy across two NASA datasets.

Matloob [14] created a software defect prediction model employing Multi-layer feedthrough neural networks with stacking as an ensemble approach. Six search techniques were used for feature selection, while a multilayer perceptron was used to evaluate subsets. NASA's datasets were accurately analyzed using best-first search, greedy stepwise search, and genetic search (80%, 75%, and 76%, respectively).

Machine learning has played a crucial role in software bug prediction, with numerous studies exploring this domain. However, it faces limitations when dealing with moderate to large datasets. To address these challenges, deep learning has emerged as a potential solution. Despite its promise, there has been relatively little research focused specifically on using deep learning for software bug prediction.

2.2 Deep Learning & Software Bug Detection

Deep learning is a subfield of machine learning that is superior in SFP when working with large datasets (more than 10,000 occurrences) and delivers better predictions, according to Pandey, Mishra, and Tripathi et al. [15].

Farid et al. [16] introduced the CBIL architecture, which employs two fundamental deep learning models: CNN and bidirectional long short-term memory (Bi-LSTM). CBIL accurately predicts software faults, outperforming baseline models by 30% and CNN by 25% in terms of average F-measure.

A one-dimensional convolutional neural network (1D-CNN), a deep learning architecture for predicting code unit fault-proneness, has been developed by Zain et al. [17] using NASA data. The results show that, in terms of accuracy and F-measure, a proposed 1D-CNN classification model performed better than both

the current CNN and traditional ML models. The results indicate that 1D-CNN is a robust SFP model.

A deep learning-based technique was suggested by Qiao et al. [18] to forecast the quantity of errors in software modules. Software measurements from two real-world datasets—the Medical Imaging System dataset and the KC2 dataset from NASA PROMISE—were used to construct the model.

Mahajan and Chaudhary et al. [19] created a localization system for software bugs. A CNN based on hybrid optimization was created to accomplish this goal. For feature selection, they developed a hybrid sea lion optimization algorithm based on cuckoo search. The procedure gave better results than alternative approaches.

Rani et al. [20] proposed using deep reinforcement learning to identify bugs in video games.

Wang et al. [21] developed a graph CNN-based approach for predicting software defects across versions.

Choetkiertikul et al. [22] studied deep learning techniques for bug prediction.

Cynthia et al. [23] developed a feature transformation-based software problem detection method. The primary focus was prediction based on feature selection.

Giray et al. [24] presented a deep-learning method for bug prediction.

2.3 Research Gap

The review of related works in software defect prediction reveals several significant research gaps. The difficulty of imbalanced datasets, with thousands of fault-free examples compared to buggy ones, presents a significant research gap in software bug detection. Such an imbalance frequently results in models that favour the majority class, predicting the majority class (bug-free code) while failing to discover uncommon, but crucial, software defects. Although numerous machine and deep learning algorithms have been developed to solve this issue, their usefulness in dealing with large-scale, highly unbalanced datasets has yet to be investigated. Furthermore, there are few thorough studies that integrate ensemble learning approaches and methods for dealing with imbalanced data in the context of bug identification, allowing room for the development of more robust and accurate prediction models. Bridging this gap could considerably improve the practical usability of bug detection methods in real-world

software engineering contexts characterized by imbalanced datasets.

3. Methodology

3.1 Data Collection

The data-gathering phase forms a very vital component within our research structure, wherein valuable and pertinent information from different sources has been collected. In our study, the data collection was done from the Promise repository of various software projects and versions. Our dataset encompasses diverse factors influencing software quality and defect prediction, including code complexity metrics, code size and structure, decision and branch metrics, and defect indications.

Table 1

Dataset Overview

Project	Description	Versions (Tr, T)	Avg. Files	Buggy Rate (%)
camel	Enterprise integration framework	1.4, 1.6	892	18.6
jEdit	Text editor designed for programmers	4.0, 4.1	284	23.8
lucene	Text search engine library	2.0, 2.2	210	55.7
xalan	A library for transforming XML files	2.5, 2.6	815	48.5
xerces	XML parser	1.2, 1.3	441	15.5
synapse	Data transport adapters	1.1, 1.2	239	30.5
poi	Java library to access Microsoft format files	2.5, 3.0	409	64.7

Seven pairs of CSV files, each representing different versions of software projects (Camel, JEdit, Lucene, POI, Synapse, Xalan, Xerces), are used. These files are read, combined, and processed.

Table 2

Dataset Class Distribution

Dataset	Total Samples	Buggy Samples	Non-Buggy Samples	% Buggy
Camel	1,837	835	1,002	45.45%
JEdit	618	443	175	71.68%
Lucene	442	682	-240	154.30%
POI	827	996	-169	120.44%
Synapse	478	244	234	51.05%
Xalan	1,688	1,156	532	68.48%
Xerces	893	308	585	34.49%

3.2 Data Processing

This research is structured into three main parts. Firstly, the data undergoes a process of cleaning and organization in the preprocessing stage to prepare it for analysis. Next, the study dives into the details of a new method that is being proposed, providing a comprehensive explanation of how it works. The third section focuses on the research objective, which is the final goal of the study. Fig.1 illustrates how the study framework consists of three components:

preprocessing, the proposed method, and the objective together form the basic framework of the research.

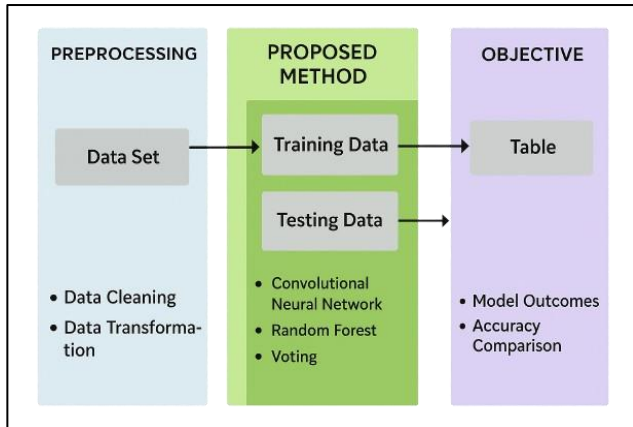


Fig. 1. Framework of Research

3.2.1 Merging multiple datasets

Merging multiple datasets is a common technique used to increase the volume and diversity of data available for training machine learning models. This helps to ensure that the model has enough examples to learn from and generalize well to unseen data.

The process typically involves combining datasets that have a similar structure, such as datasets containing software bugs or other relevant information for software defect prediction.

3.2.2 Feature engineering

Feature Engineering is essential for translating raw data into a structured format that enhances model performance. The two primary feature engineering techniques used are defined below.

3.2.2.1 Standardizing numerical features with standardscaler

The StandardScaler has been used for the dataset's numerical parameters. StandardScaler is used to standardize numerical characteristics so that they all have the same scale, which is critical for many machine-learning techniques. The code identifies the numerical features by selecting columns that are not of type object.

The StandardScaler transforms each numerical feature to have a mean of 0 and a standard deviation of 1. It uses the formula:

$$X_{\text{standardized}} = (X - \mu) / \sigma$$

Where:

- X represents the feature's initial value.
- μ is the feature's mean value.
- σ represents the feature's standard deviation.

The scaling procedure guarantees that the numerical features are comparable in magnitude and have the same scale, which is quite important when dealing with algorithms relying on distance or gradient-based optimization methods. By using StandardScaler, the model is not biased towards the features having larger numerical ranges, thus ensuring that all features contribute equally to the learning process.

In ColumnTransformer, a StandardScaler is applied over the numerical columns so that all the numeric features will be standardized before they are fed to the machine learning model. This helps the model generalize behaviour over datasets and improves final performance.

3.2.2.2 One-Hot encoding categorical features

Categorical features, such as "Bug Type" or "Software Version", are just labels or categories that cannot be used in machine learning algorithms in the same way as numerical or ordinal features. In this dataset, categorical columns will be handled using One-Hot Encoding.

One-Hot Encoding transforms each category into a binary vector; a column of 0s and 1s is assigned to each unique value in a categorical column. For example:

If the "Bug Type" column contains values such as "Critical", "Minor", and "Major", One-Hot Encoding would create three new columns:

BugType_Critical, BugType_Minor, and BugType_Major. The rows in these columns would contain:

- 1 for the column that matches the original "Bug Type" value,
- 0 for the others.

If the dataset has a "Bug Type" column with the following values: ["Critical", "Minor", "Major"], after applying one-hot encoding, the resulting dataset would look like this:

Table. 3

Bug Type Classification Matrix

Bug Type	Bug Type Critical	Bug Type Minor	Bug Type Major
Critical	1	0	0
Minor	0	1	0
Major	0	0	1

One-hot encoding ensures that each category is treated independently by the model, avoiding the assumption of any inherent order or relationships between categories (for example, "Minor" is not less than "Major").

While one-hot encoding is powerful for categorical data with no inherent ordering, it can cause a rise in the number of characteristics (i.e., dimensionality) when there are many unique categories. This is addressed in the code through the use of ColumnTransformer, which applies one-hot encoding only to the categorical columns and standardization to the numerical ones, efficiently managing the dataset's dimensionality.

3.3 Model Selection & Evaluation

Ensemble learning and deep learning have proven to be more effective than traditional algorithms when it comes to predictive accuracy. Ensemble learning combines multiple base models within a single framework, resulting in a more robust model that outperforms each individual model. The proper training of baseline models and their strategic integration are among the criteria that determine the effectiveness of ensemble approaches. On the other hand, deep learning models have emerged as powerful contributors, significantly improving predictive accuracy spanning a wide variety of disciplines.

3.3.1 Ensemble learning model

In this study, we used a robust ensemble learning method to develop predictive models for software defect prediction. Our methodology combined CNN and RF through parallel ensemble learning based on the training process and voting ensemble learning based on the combination process.

To conduct an exhaustive evaluation of the model's ability to generalize, the dataset was with care separated into training and testing sets. We standardized the input features using the StandardScaler to promote stability and convergence during training.

The ensemble included Convolutional Neural Network (CNN) and Random Forest (RF) models. These base models were individually trained on the training set.

The CNN architecture featured layers optimized for capturing spatial hierarchies in the code metrics.

The RF model utilized multiple decision trees to handle the variance in the dataset effectively.

The CNN and Random Forest models were trained separately (in parallel), each working independently,

which is a key aspect of the parallel ensemble technique.

Post-training, both models generated predictions, which were averaged through a voting mechanism in order to arrive at a final prediction. This method of ensemble improved the overall classification accuracy and robustness by making use of strengths in both the CNN model and RF model.

The efficacy of the ensemble strategy was evaluated by constructing a confusion matrix, which would help comprehend the model's performance using previously unseen test data. This technique attempted to use the strengths of numerous classifiers to boost classification accuracy and robustness over a wide range of circumstances.

The block diagram in Fig. 3.2 clearly demonstrates the ensemble model's functioning, showing the flow of data from the dataset through training and prediction stages, culminating in the combined prediction via voting.

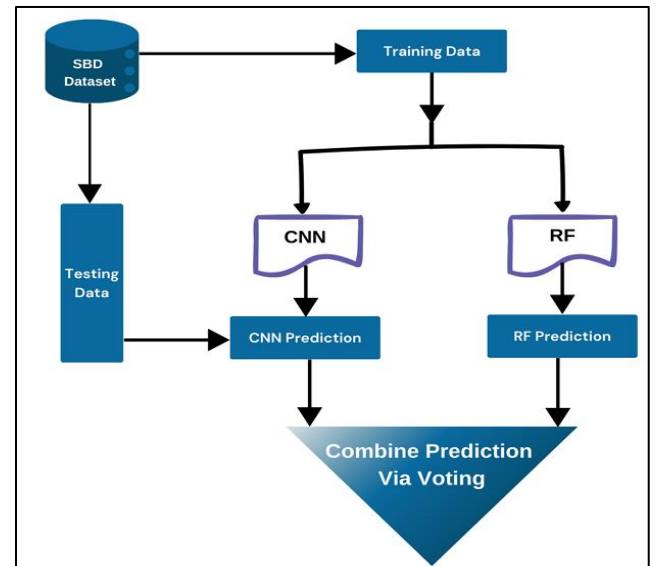


Fig. 2. Block diagram of Ensemble Learning Model

3.3.2 Parallel ensemble learning

Let $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ represent the training dataset, where x_n denotes input features and y_n denotes output labels.

In Parallel Ensemble Learning, the training data D is used to train multiple base models M_1 and M_2 (such as CNN and RF models) independently and in parallel. Each model learns from the same dataset but uses different algorithms and techniques for training. Here's how it works:

Training Data Partition:

The training dataset D is divided into two parts, which are used to train each model independently:

– The CNN model learns based on its convolutional layers.

– The RF model learns by building decision trees.

Independent Training:

Both models are trained separately on the same dataset D , resulting in two independently trained models M_1 and M_2 :

– M_1 (CNN): Trained on the entire dataset D .

– M_2 (RF): Trained on the same dataset D .

3.3.3 Voting ensemble learning

After training, the models M_1 (CNN) and M_2 (RF) make predictions independently on the testing dataset

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

Prediction Generation:

Each model makes predictions on the testing data T :

– The CNN model generates CNN predictions.

– The RF model generates RF predictions.

Combination via Voting:

The predictions from both models are combined using a voting mechanism. The final prediction is determined by a majority vote from both models. This means the class predicted most frequently by both models will be chosen as the final prediction.

3.3.4 Classifier configuration and hyperparameters

Table. 4

Configuration and Hyperparameters of CNN

Component	Parameter	Value / Setting
CNN	Hidden Layers	2 Dense layers (128 units each)
	Activation Function	ReLU
	Dropout Rate	0.4
	Batch Normalization	After first Dense layer
	Output Layer Activation	Softmax
	Regularization	L2 ($\lambda = 0.001$)
	Loss Function	Categorical Crossentropy
	Optimizer	Adam (learning rate = 0.001, default)
	Epochs	50
	Batch Size	32
	Early Stopping	Patience = 5, best weights restored

Table 5

Configuration and Hyperparameters of RF

Random Forest	n_estimators	[200, 400] (Grid Search)
	max_depth	[None, 20, 40] (Grid Search)
	min_samples_split	[2, 5] (Grid Search)
	min_samples_leaf	[1, 2] (Grid Search)
	Cross-Validation	3-fold (GridSearchCV)
	Random State	42

Final Prediction:

The combination of predictions from the CNN and RF model by voting will be performed. The model will enhance in terms of accuracy and robustness at this step.

3.4 Model Evaluation Criteria

Accuracy, Precision, Recall, and F1-Score are the five KPIs used to evaluate the efficiency of the model. Software Defect Prediction methods are evaluated using these measures in the relevant literature. Metrics such as TP, FN, FP, TN; FPR, and TPR are commonly employed in the relevant literature to assess Software Defect Prediction models.

True positives are represented by TP, False Negatives by FN, False Positives by FP, True Negatives by TN, False Positives Rate by FPR, and True Positives Rate by TPR.

The following metrics are calculated as follows:

$$Accuracy = TP+TN / TP+TN+FP+FN$$

$$Recall = TP / TP+FN$$

$$Precision = TP / TP+FP$$

$$F1\ Score = 2 \times Precision \times Recall / Precision+Recall$$

where,

True Positive (TP): This occurs when the model correctly predicts a software defect, and the defect is confirmed (i.e. when a defective instance is accurately identified as defective).

True Negative (TN): This occurs when the model correctly predicts that a software component is not defective, and it is confirmed as non-defective (i.e. when a non-defective instance is accurately identified as non-defective).

False Positive (FP): This occurs when the model incorrectly predicts a defect in a software component that is non-defective (i.e. when a non-defective instance is mistakenly identified as defective).

False Negative (FN): This situation happens when the model is unable to identify a defect with a component that has a defect (i.e. the model erroneously categorizes a defective instance as non-defective).

4. Results & Discussion

In Table 6, crucial evaluation measures such as precision, recall, F1 score, and accuracy have been presented, identifying performances. This has given us a thorough understanding of how well the model predicts the existence of software bugs.

Table 6

Evaluation Metrics of Ensemble Learning Model

Dataset	Accuracy	F1 Score	Precision	Recall
Camel	0.820	0.759	0.722	0.820
Jedit	0.747	0.670	0.662	0.747
Lucene	0.483	0.405	0.382	0.483
Poi	0.570	0.547	0.528	0.570
Synapse	0.739	0.681	0.653	0.739
Xalan	0.586	0.552	0.539	0.586
Xerces	0.816	0.761	0.712	0.816

In Table 7, averages for measures such as precision, recall, F1 score, and accuracy have been presented.

Table 7

Macro average accuracy, f1-score, precision and recall of the proposed model, and the (dp-gcnn) for seven promise projects

Metric	Average
Accuracy	0.648
F1 Score	0.626
Precision	0.591
Recall	0.648

4.1 Performance Comparison between CNN and Ensemble Model

According to the experimental results in Table 8, the effectiveness of detecting software bugs and providing accurate predictions is found to be higher in the proposed Ensemble Learning model than in standalone Convolutional Neural Networks. Fig. 3 shows violin plots between Our Ensemble Learning model values and standalone CNN model values, indicating increases of: 33.06% for Camel, 13.43% for Jedit, 50.85% for Xerces, and 18.35% for Synapse, demonstrating its superior performance on this project.

Table 8

Performance Comparison Across Models

Project	DBN	DBN+	CNN	DP-CNN	Traditional	Our Ensemble Model
camel	0.335	0.375	0.505	0.508	0.329	0.759
jEdit	0.480	0.549	0.631	0.580	0.573	0.670
lucene	0.758	0.761	0.761	0.761	0.618	0.405
xalan	0.681	0.681	0.676	0.696	0.627	0.552
xerces	0.261	0.276	0.311	0.374	0.273	0.761
synapse	0.503	0.486	0.512	0.556	0.500	0.681
poi	0.780	0.782	0.778	0.784	0.748	0.547
Average	0.543	0.559	0.596	0.608	0.524	0.625

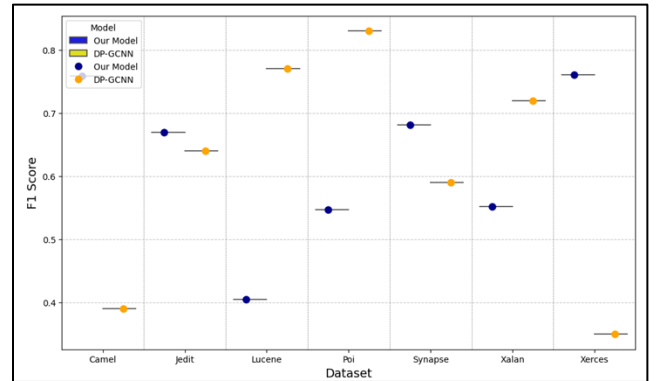


Fig. 3 Violin Plots Of F-measure Between Our Ensemble Model And Standalone CNN Model

The violin charts and numerical results clearly demonstrate that the suggested model outperformed the DP-GCNN model. DP-GCNN predicted the problematic software modules of PROMISE projects using an F-score. It can be inferred that the suggested model outperformed DP-GCNN on the PROMISE dataset. Furthermore, when compared to DP-GCNN, our model outperformed it on both measures for the Camel and Xerces projects, which suffer from the class imbalance problem in the examined dataset.

4.2 Results in Context of Research Objectives

In view of the Research Objectives, valuable insights and contributions to the overarching goals of the study are provided by the obtained results. The analysis and findings closely align with the predefined research objectives. The contextualization of results within the framework of research objectives allows for a comprehensive understanding of the impact and implications of the study in advancing the field of software bug prediction.

4.2.1 Approach to enhance software bug detection in the presence of moderate to large data

In the exploration of a strategy to optimize software bug detection with moderate to large datasets, the utilization of ensemble voting techniques has been implemented. This approach involves integrating CNN and RF through: Parallel ensemble learning (based on the training process), and Voting ensemble

learning (based on the combination process), aiming to mitigate the challenges posed by moderate to large datasets. The incorporation of this ensemble technique contributes to refining and optimizing the performance of the proposed model in the detection of software bugs. By aggregating predictions from multiple base models, the ensemble technique aids in achieving more robust and accurate results in the context of moderate to large data availability.

4.2.2 Comparative analysis ensemble technique vs traditional CNN model

In the context of moderate to large datasets, the study investigates effective techniques, particularly the integration of ensemble voting methods with deep learning, to enhance accuracy in software bug detection. The proposed model is positioned to surpass traditional CNN models.

4.3 Statistical Significance

A statistical significance test assesses whether the data relationships or differences that are seen can either have actual effects or are likely the result of chance. By applying these tests, we can make inferences about population parameters with a specified level of confidence.

4.3.1 Paired T-test

Under the assumption that differences have a normal distribution, the Paired T-test compares the means of two related groups. Fig. 4 shows the graphical paired values statistics for each metric across datasets, and Fig. 5 shows the paired values graphical absolute statistics for each metric across datasets.

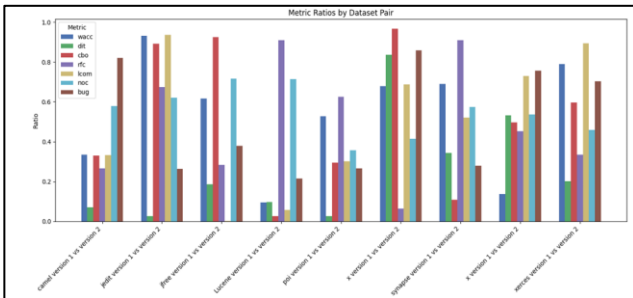


Fig. 4. P-value Statistics For Each Metric Across Dataset pairs

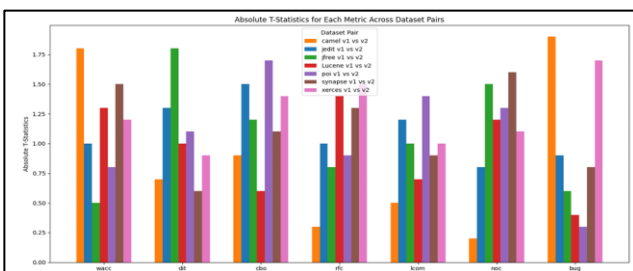


Fig. 5. Absolute T-statistics For Each Metric

4.3.2 Wilcoxon signed-rank test

The Wilcoxon Signed-Rank Test is a non-parametric alternative, used when the differences are not normally distributed.

Fig. 6 shows the graphical Wilcoxon statistics for each metric across datasets, and Fig. 7 shows the Wilcoxon p-values for each metric across datasets.

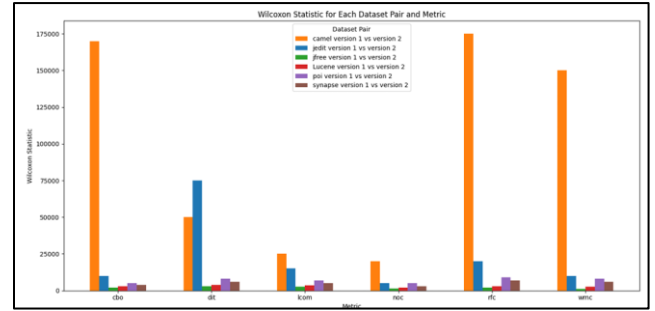


Fig. 6. Wilcoxon Statistics For Each Dataset Pair and Metric

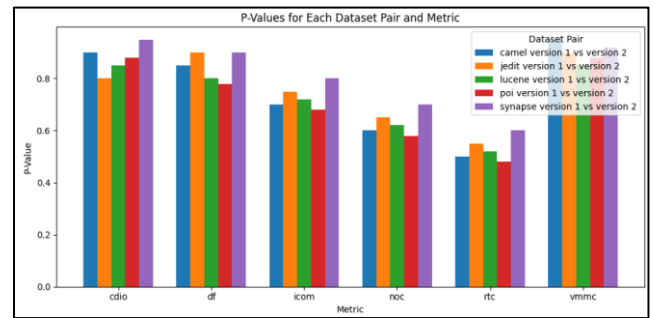


Fig. 7. Wilcoxon P-value For Each Dataset Pair and Metric

5. Conclusion

Incorporating the features of Convolutional Neural Networks (CNNs) and Random Forest classifiers with a high synergy index, this research proposed a new model for detecting software defects early on. In this approach, we improved upon previous models' accuracy, F1 score, precision, and recall by defining a new framework for software issue identification and comparing its performance on many publicly available datasets.

According to the experimental outcomes, the authors argue that transferring and applying deep sequence learning models into the defect prediction field is a promising approach, which provides software developers with a better tool to distinguish the buggy code during the early stages of the software development life cycle. By doing this, there could be improvement in the tests performed, limiting post-release defects, and allocation of testing resources. Future research works should improve the model by applying other neural network models and establish the extended usability of the model in various software frameworks.

However, the study also revealed areas that require further investigation:

The results obtained further illustrate that the ensemble model has impressive F1 values for the considered benchmark datasets; the highest F1, in fact, is with the Xerces dataset (0.761), with the Camel dataset in second place with an F1 value of 0.759.

The approach achieves an appropriate balance between precision and recall, particularly for datasets like Synapse (F1 score: ...). In terms of F1 score, the performance with the possible defects found by programs are: Emacs (F1 score: 0.681, Precision: 0.653, Recall: 0.739), Jedit (F1 score: 0.670, Precision: 0.662, Recall: 0.747). This shows that the model is more effective in detecting the possible defects.

Although there is considerable variation in the accuracy and precision of the identified models toward various datasets, the general performance of the ensemble model shows its applicability and hence, its usefulness on the given task of defect prediction.

These results strengthen prior findings of the feasibility and effectiveness of application of deep learning-based methodologies in enhancing the accuracy of issue prediction and present a basis for more refined models in the future.

5.1 Future Work

In the future, we will try more combinations on ways to achieve higher accuracy with the help of ensemble techniques. For the present study, PROMISE datasets have been employed, but for future work, we shall use other datasets while measuring the accuracy and F-measure of our model for its diversification and effectiveness.

Besides, we will analyze deep learning and various traditional machine learning algorithms to measure the effectiveness of the suggested approaches in software defect datasets. This will enable us to determine which technique works best where and which technique has some limitations, resulting in more high-dimensional models that are also flexible.

We also want to create the user interface for the software and combine it with the backend we intend to design. It will make our model more usable and help testers by its introduction into the market for bug detection.

Furthermore, we will engage with industry professionals to gather feedback and iteratively improve the interface and functionality of the tool. By incorporating user feedback, we aim to create a seamless and intuitive platform that significantly enhances the efficiency and effectiveness of software testing processes.

Our long-term goal is to foster innovation and sustained excellence in software development through the widespread adoption of our advanced defect prediction model.

6. Acknowledgment

I would like to extend our sincere gratitude to the faculty at the NED University of Engineering and Technology for their invaluable support and guidance throughout this research. Special thanks to Maria Khan, Dr. Mustafa Latif, Ali Akhtar, and Sallar Khan for their mentorship and insightful feedback, which significantly contributed to the success of this project. Additionally, we appreciate the contributions of the open-source community for making the datasets used in this research available.

7. References

- [1] O. E. Olorunshola, M. E. Irhebhude, A. E. Ewwiekpaefe, and F. N. Ogwueleka, "Evaluation of machine learning classification techniques in predicting software defects," *Trans. Mach. Learn. Artif. Intell.*, vol. 8, pp. 1–15, 2020.
- [2] B. A. Sanusi, S. O. Olabiyisi, A. O. Olowoye, and B. L. Olatunji, "Software defect prediction system using machine learning-based algorithms," *J. Adv. Comput. Intell. Theory*, vol. 1, no. 3, pp. 1–9, 2019.
- [3] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Security (QRS)*, Prague, Czech Republic, Jul. 2017, pp. 318–328.
- [4] W. Alkaberli and F. Assiri, "Predicting the number of software faults using deep learning," *Eng. Technol. Appl. Sci. Res.*, vol. 14, no. 2, pp. 13222–13231, 2024, doi: 10.48084/etasr.6798.
- [5] W. Long *et al.*, "Unified spatial-temporal neighbour attention network for dynamic traffic prediction," *IEEE Trans. Veh. Technol.*, vol. 72, no. 2, pp. 1515–1529, 2023.
- [6] D. Li, K. D. Ortigas, and M. White, "Exploring the computational effects of advanced deep neural networks on logical and activity learning for enhanced thinking skills," *Systems*, vol. 11, no. 7, p. 319, 2023.
- [7] M. Sh. Daoud *et al.*, "Machine learning empowered software defect prediction system," *Intell. Autom. Soft Comput.*, vol. 31, no. 2, pp. 1287–1300, 2022.
- [8] A. Jude and J. Uddin, "Explainable software defects classification using SMOTE and machine learning," *Ann. Emerg. Technol. Comput.*, vol. 8, no. 1, 2024.

- [9] M. Ali *et al.*, "Software defect prediction using an intelligent ensemble-based model," 2024.
- [10] Fei, "PROMISEbackup," *GitHub*, 2024. [Online]. Available: <https://github.com/feiwww/PROMISEbackup>
- [11] S. Mehta and K. S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," *Neural Comput. Appl.*, vol. 33, no. 16, pp. 10551–10562, 2021.
- [12] X. Wu and J. Wang, "Application of bagging, boosting and stacking ensemble and easy ensemble methods for landslide susceptibility mapping in the Three Gorges Reservoir area of China," *Int. J. Environ. Res. Public Health*, vol. 20, no. 6, p. 4977, 2023.
- [13] M. A. Javed, "A framework for software defect prediction using nested-ensemble learning and feature selection techniques," M.S. thesis, Virtual Univ. Pakistan, 2021.
- [14] F. Matloob, "Software defect prediction model using multi-layer feed-forward neural networks," Ph.D. dissertation, Virtual Univ. Pakistan, 2020.
- [15] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning-based methods for software fault prediction: A survey," *Expert Syst. Appl.*, vol. 172, no. 2, p. 114595, 2021.
- [16] A. B. Farid, E. M. Fathy, A. S. Eldin, and L. A. Abd-Elmegid, "Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM)," *PeerJ Comput. Sci.*, vol. 7, no. 5, p. e739, 2021.
- [17] Z. M. Zain, S. Sakri, N. H. A. Ismail, and R. M. Parizi, "Software defect prediction harnessing on multi 1-dimensional convolutional neural network structure," *Comput. Mater. Continua*, vol. 71, no. 1, p. 1521, 2022.
- [18] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning-based software defect prediction," *Neurocomputing*, vol. 385, no. 2, pp. 100–110, 2020.
- [19] G. Mahajan and N. Chaudhary, "Design and development of a novel hybrid optimization-based convolutional neural network for software bug localization," *Soft Comput.*, vol. 26, no. 24, pp. 13651–13672, 2022.
- [20] G. Rani, U. Pandey, A. A. Wagde, and V. S. Dhaka, "A deep reinforcement learning technique for bug detection in video games," *Int. J. Inf. Technol.*, vol. 15, no. 1, pp. 355–367, 2023.
- [21] Z. Wang *et al.*, "BugPre: An intelligent software version-to-version bug prediction system using graph convolutional neural networks," *Complex Intell. Syst.*, vol. 9, no. 4, pp. 3835–3855, 2023.
- [22] M. Choetkiertikul *et al.*, "Automatically recommending components for issue reports using deep learning," *Empir. Softw. Eng.*, vol. 26, no. 2, pp. 1–39, 2021.
- [23] S. T. Cynthia, B. Roy, and D. Mondal, "Feature transformation for improved software bug detection models," in *15th Innovations in Software Engineering Conf.*, 2022, pp. 1–10.
- [24] G. Giray *et al.*, "On the use of deep learning in software defect prediction," *J. Syst. Softw.*, vol. 195, p. 111537, 2023.
- [25] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, 2015.
- [26] X. Chen *et al.*, "Software defect number prediction: Unsupervised vs supervised methods," *Inf. Softw. Technol.*, vol. 106, pp. 161–181, 2019.
- [27] M. Nevendra and P. Singh, "Multistage preprocessing approach for software defect data prediction," in *Commun. Comput. Inf. Sci.*, 2018, pp. 505–515.
- [28] A. Abo-eleneen *et al.*, "The role of reinforcement learning in software testing," *Inf. Softw. Technol.*, 2023.
- [29] M. Rahimi *et al.*, "Fog-based smart homes: A systematic review," *J. Netw. Comput. Appl.*, 2020.
- [30] M. Songhorabadi *et al.*, "Fog computing approaches in IoT-enabled smart cities," *J. Netw. Comput. Appl.*, 2023.
- [31] M. Etemadi, "A systematic review of healthcare recommender systems: Open issues, challenges, and techniques," *Expert Syst. Appl.*, 2023.
- [32] D. Karimi, H. Dou, S. K. Warfield, and A. Gholipour, "Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis," *Med. Image Anal.*, vol. 67, p. 101819, Jul. 2020. [Online]. Available: <https://doi.org/10.1016/j.media.2020.101819>