Use of Deep Learning in Early Software Bug Detection

Syed Mibran Hassan Zaidi* and Mustafa Latif

Department of Computer Science and Information Technology, NED University of Engineering and Technology, Karachi, Pakistan

ABSTRACT Traditional defect prediction studies primarily rely on hand-crafted features fed into Machine Learning (ML) classifiers to identify defective code. However, these features frequently fail to capture the essential semantic and structural information of programs and metric-based datasets, which are critical for accurate defect prediction. To address these limitations, the current study introduced a comprehensive deep learning pipeline for Software Defect Prediction (SDP) using multiple publicly-available datasets. The study particularly focused on the integration of Convolutional Neural Networks (CNN) and Random Forest classifiers. The datasets, representing different versions of software projects, were loaded, concatenated, and preprocessed using a combination of StandardScaler and OneHotEncoder. This preprocessing ensures the data is in a suitable format for training models. The approach adopted by the current study involved building and training a CNN model to capture semantic and structural features of the software data, followed by a Random Forest model tuned through GridSearchCV for optimal performance. Predictions from both models were combined using an ensemble method, where a majority vote determined the final predictions. The accuracy, precision, recall, and F1 score of this ensemble model were calculated to evaluate its performance. The experimental results demonstrated that the ensemble model, leveraging the strengths of both CNN and Random Forest classifiers, achieved high accuracy and F1 scores across multiple datasets. This study highlighted the effectiveness of combining deep learning with traditional ML techniques for SDP. This offered a robust framework to improve software reliability and aid developers in order to identify potential bugs.

INDEX TERMS Convolutional Neural Networks (CNN), deep learning, early Software Defect Prediction (SDP), Random Forest

I. INTRODUCTION

Machine Learning (ML) techniques have become integral in analyzing the data from diverse perspectives. This offers critical insights to the developers that enhance software development processes. These techniques have shown particular promise in the realm of software bug prediction, a crucial aspect of ensuring software reliability.

Effective Software Defect Prediction (SDP) is essential to improve software reliability by helping developers identify potential bugs and allocate testing resources more efficiently. Traditional defect prediction methods typically rely on hand-crafted features that are entered into ML classifiers to detect defective code. However, these hand-crafted features often fall short in capturing the essential semantic and structural information of programs, as well as metric-based datasets, which are vital for accurate defect prediction.

To address these limitations, the current study proposed a novel deep learning

*Corresponding Author: mibranhassan@gmail.com pipeline designed for SDP, utilizing multiple publicly-available datasets. The approach involved a detailed evaluation of various ML techniques, with a particular focus on integrating Convolutional Neural Networks (CNN) and Random Forest classifiers. The datasets, representing different versions of software projects, were carefully loaded, concatenated, and preprocessed using StandardScaler and OneHotEncoder to prepare them for model training.

The study built and trained a CNN model to automatically extract semantic and structural features from the software data and optimize a Random Forest model through GridSearchCV. The predictions from these models were combined using an ensemble method, where a majority vote determined the final output. The performance of this ensemble approach was assessed based on accuracy and F1 score. The results demonstrated that the ensemble model, which leverages the strengths of both CNN and Random Forest classifiers, achieved high performance across multiple datasets.

This research used deep learning to predict software defects early, enhancing the overall software quality and resource allocation. It pioneers proactive defect detection, optimizes testing efforts, and integrates deep learning for informed decision-making in software development, fostering innovation and sustained excellence. The ultimate goal was to revolutionize the field by seamlessly incorporating deep learning into software development processes.

A. RESEARCH OBJECTIVES

The current study aimed to address the following research objectives:

- Develop and implement deep learning models that enable the early detection of software defects during the development lifecycle, minimizing the impact on downstream processes.
- Optimize testing efforts by leveraging deep learning algorithms to identify and prioritize areas of the codebase more prone to defects, allowing for more targeted testing.
- Contribute to the optimization of resource allocation in software development by efficiently identifying and addressing software defects early in the development lifecycle.

B. RESEARCH QUESTIONS

The research sought to address the following questions:

RQ1: How effective is the proposed approach utilizing enhanced CNNs in software bug detection compared to traditional defect prediction models?

RQ2: To what extent does the integration of deep learning techniques, specifically CNNs, contribute to the accuracy and efficiency of software bug prediction in comparison to conventional ML methods?

RQ3: How does the proposed approach perform in predicting defects across diverse open-source software defect datasets, and what insights can be drawn from its performance in real-world scenarios?

In developing advanced software defect detection through deep learning, ethical considerations are paramount. Transparency, privacy protection, and bias mitigation are crucial in handling open-source software defect datasets responsibly. Adherence to ethical guidelines in data collection, respecting consent and anonymization, is vital. The choice of evaluation metrics should prioritize fairness, avoiding unintended consequences. A rigorous ethical scrutiny of the proposed model, particularly in real-world applications, underscores the commitment to responsible innovation and user well-being.

TABLE I

SKILLS AUDIT

Skills	Skill Indicators	Initial Competence Level	Approach to Improve Competency
Programming Proficiency	Ability to write efficient code in Python	Intermediate	Engage in coding exercises, contribute to open-source projects, and take advanced programming courses.
Statistical Knowledge	Understanding of hypothesis testing and probability theory	Basic	Enroll in online courses covering statistical concepts, read relevant literature, and practice statistical analysis.
Deep Learning Algorithms	Proficiency in classification and regression algorithms	Intermediate	Explore advanced Deep Learning algorithms, participate in Kaggle competitions, and implement models on diverse datasets.
Data Preprocessing	Ability to clean, transform, and prepare datasets	Basic	Work on real-world datasets, and practice handling missing data, outliers, and normalization techniques.
Feature Engineering	Skill in identifying and engineering relevant features	Intermediate	Study feature engineering techniques, experiment with different combinations, and understand domain-specific features.
Model Evaluation Metrics	Understanding of precision, recall, F1-score, etc.	Basic	Stay updated on new metrics, analyze their implications, and implement them in model evaluation.
Domain Knowledge	Familiarity with software development processes	Intermediate	Read literature on software engineering, collaborate with industry professionals, and gain practical experience in software development.

II. LITERATURE REVIEW

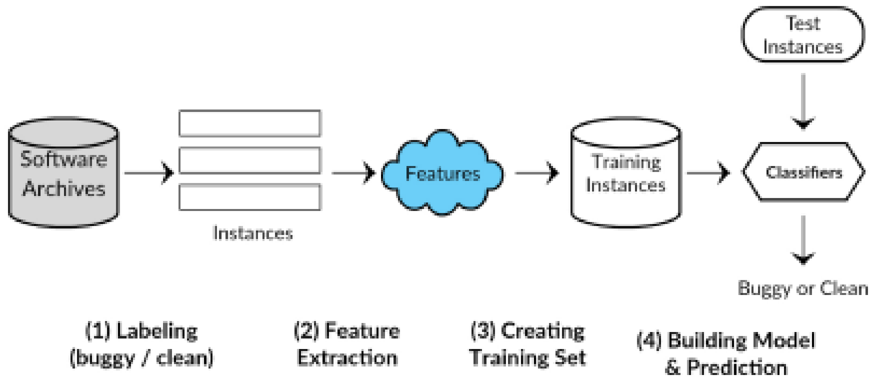

FIGURE 1. Defect prediction

SDP is a process of predicting potentially defective areas of code, which can help developers allocate their testing efforts by first checking potentially buggy code [1].

Defect prediction is essential to ensuring reliability of today's large-scale software. Figure 2 presents a typical file-level defect prediction process which is commonly adopted in the literature [2]-[4]. As the process shows, the first step is to collect source code files (instances) from software archives and label them as buggy or clean. The labeling process is based on the number of post-release defects of each file. A file is labeled as buggy if it contains at least one post-release bug. Otherwise, the file is labeled as clean. The second step is to extract features from each file. Many traditional features have been defined in past studies, which can be categorized into two kinds: code metrics (e.g., McCabe features [5] and CK features [6], and process metrics (e.g., change histories). The instances with the corresponding features and labels are subsequently employed to train predictive classifiers using various ML algorithms, such as SVM, Naive Bayes, and dictionary learning [2]. Finally, new instances are fed into the trained classifier, which may predict whether the files are buggy or clean. The set of instances used for building the classifier is training set, while test set includes the instances used for evaluating the learned classifier. The current study focused on within-project defect prediction, that is, the training and test sets 319 belong to the same project. Following the previous work in this field [3], the current study used instances from an older version of this project for training, and instances from a newer version for test.

CNNs are a specialized kind of neural networks for processing that data that have a known grid-like topology [7], such as time-series data in 1D grid and image data in 2D grid. CNNs have been demonstrated successful in many practical fields, including speech recognition [8], image classification [9], [10] and Natural Language Processing (NLP) [11].

SDP has garnered substantial attention in recent years due to its critical role in improving software quality and reliability. Various methodologies have been explored to enhance the accuracy and efficiency of defect prediction.

ML has played a crucial role in software bug prediction, with numerous studies exploring this domain. However, it faces limitations when dealing with large and complex 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.

[12] introduced a new convolutional graph neural network (GCNN) which leverages the Abstract Syntax Tree (AST) of source code for defect prediction, adapting to various project sizes. This model outperforms traditional approaches in AUC and F-score, offering strong predictive capabilities. [13]-[16] introduced a framework for benchmarking classification algorithms on software defect datasets. This concluded that while their selected algorithms offered good prediction accuracy, there was no significant difference in performance among the various classifiers. This study, however, did not cover the full spectrum of machine learning techniques available for defect prediction. Sharma and Jain [17] utilized the WEKA tool to compare different classification algorithms but did not specifically apply these methods to software bug prediction. Similarly, Kaur and Pallavi [18] examined various data mining techniques for defect prediction without providing a comparative performance analysis. Wang et al. [19]

focused exclusively on ensemble classifiers, highlighting their potential but not considering other ML methods. These studies underscore a gap in comprehensive comparative analyses of diverse ML techniques in the context of software defect prediction.

Historically, research has also delved into the use of software metrics for defect prediction. Chidamber and Kemerer [6] demonstrated the suitability of software metrics for defect prediction, while Basili et al. [20] validated object-oriented design metrics (OODM) for this purpose. Gyimothy et al. [21] advanced this field by using object-oriented metrics analysis to enhance prediction performance. Singh and Verma [22] argued for the use of design metrics during the software development lifecycle to enable early defect detection and cost-effective development.

In exploring classification methods, [23] proposed a polynomial function-based neural network (pf-NN) model, combining fuzzy C-means and genetic clustering techniques to handle nonlinear parameters. Despite these advances, clustering techniques alone may not achieve optimal prediction performance. Various ML algorithms, such as Naïve Bayes (NB), Support Vector Machines (SVM), and Decision Trees (DTs), have been employed to overcome these limitations. Elish et al. [24] found that SVMs were highly effective in defect detection as compared to other models. Similarly, Shivaji et al. [25] demonstrated that feature selection methods could enhance NB classifiers' performance. Dejaeger et al. [26] highlighted that augmented NB classifiers outperformed other models in terms of ROC curve analysis. DT approaches, as explored by [27], also showed promise, while Singh and Verma [28] demonstrated the effectiveness of multi-classifier approaches combining SVM, NB, and Random Forest (RF). Singh et al. [29] further explored fuzzy rule-based techniques, showing competitive results as compared to traditional methods.

Recently, Yang et al. [30] applied deep learning techniques to defect prediction, utilizing deep belief networks on large open-source projects to achieve significant improvements in prediction accuracy.

III. METHODOLOGY

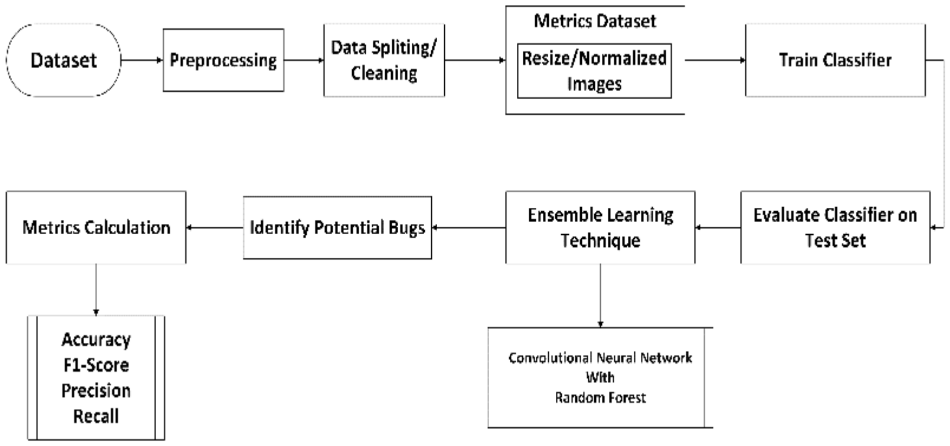

FIGURE 2. Proposed model

The code involves a comprehensive deep learning pipeline that processes multiple datasets, trains models, and evaluates their performance [31]. The code uses several Python libraries for data manipulation, model building, and evaluation, including pandas, numpy, matplotlib, scikit-learn, keras, and imblearn.

TABLE II
DATASET DESCRIPTION

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), were used. These files were read, combined, and processed [32].

A. MODEL TRAINING AND EVALUATION PROCESS

A function defines a CNN model architecture. The model consists of a Conv1D layer, MaxPooling1D layer, Flatten layer, Dense layer with ReLU activation, Dropout layer, and an output Dense layer with softmax activation. The model is compiled using the Adam optimizer and the categorical inter-entropic loss. In the data preprocessing and training phase, datasets are first loaded and concatenated. Features and target variables are then separated, and categorical and numeric columns are identified. The data is preprocessed using a \`ColumnTransformer\`, which applies \`StandardScaler\` to numeric columns and \`OneHotEncoder\` to categorical columns. A pipeline is created to ensure the data is transformed into a suitable format for the CNN. The data is then divided into training and test sets. Target labels are mapped to consecutive integers and converted to a categorical format for the CNN, and the data is reshaped accordingly. For training the CNN, the model is built and trained on the training data. After training, predictions are made on the test data using the trained CNN model. In the Random Forest Classifier phase, a Random Forest model is tuned using \`GridSearchCV\` with a specified parameter grid. The best model from the grid search is selected and trained on the training data. Predictions are then made on the test data using this tuned Random Forest model.

Finally, in the ensemble predictions phase, predictions from both the CNN and Random Forest models are combined using a majority vote to form the final predictions. The accuracy and F1 score of the ensemble model are then calculated to evaluate its performance.

IV. RESULTS

The performance of the SDP model is typically evaluated using precision, recall, f-score and accuracy. To compare with the state-of-the-art models' performances, the model was evaluated using F-score. However, the study reported accuracy score as well. This is because it has a lower variance, that is, it is more static than any of the above metrics, and is therefore, highly preferable for the evaluation of defect prediction models [34].

The F-score is the harmonic mean of precision and recall. It is a widely used measure of the accuracy of the test, with values between 0 for the worst accuracy and 1 for the best accuracy.

The results of accuracy, precision, recall, and F-score performance measures are shown in Table 3, with the best results among classifiers highlighted in bold. Additionally, obtained scores are shown as the violin plots in Figure 3. Table IV shows the average accuracy, precision, recall, and F-scores obtained by the classifiers in the analyzed projects.

TABLE III
PERFORMANCE OF PROPOSED 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

TABLE IV
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

TABLE V
AVERAGE F1-SCORE OF THE PROPOSED MODEL, AND THE (DP-GCNN) FOR SEVEN PROMISE PROJECTS

Metric	Our Model	DP-GCNN
F1 Score	0.626	0.610

It can be seen from both the violin plots and numerical values that the proposed model was more successful than the DP-GCNN model. DP-GCNN's effectiveness in predicting defective software modules from PROMISE projects, measured by F-score. It can be concluded that the proposed model generally performed better than DP-GCNN for PROMISE dataset. In addition, as compared to DP-GCNN, the model performed better on both measures for camel and xerces projects, which suffer the most from the class imbalance problem in the analyzed dataset.

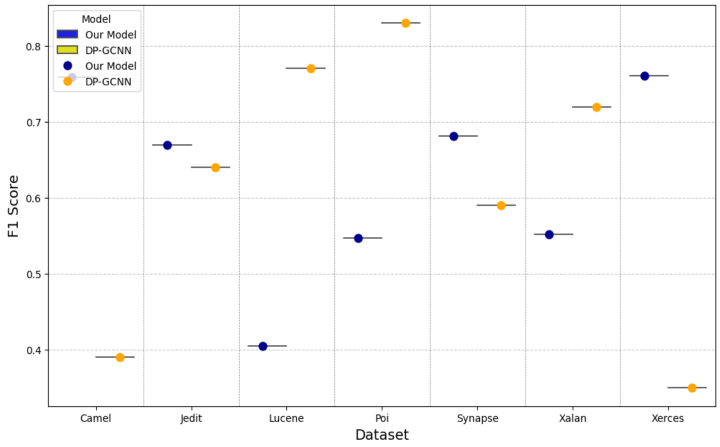

FIGURE 3. Violin plot of F-score by the proposed model and DP=GCNN model

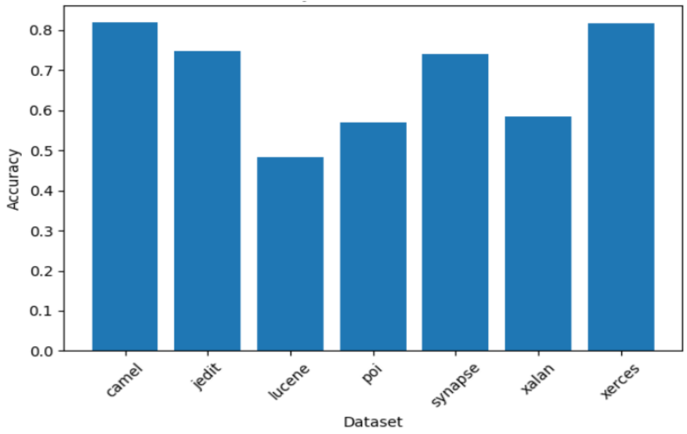

FIGURE 4. Accuracy metrics of the proposed model

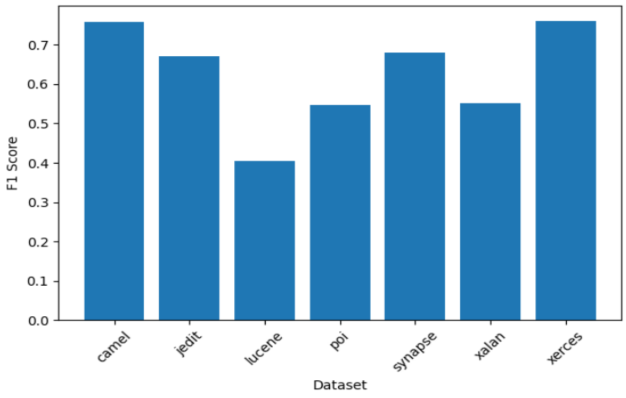

FIGURE 5.F1-score metric of the proposed model

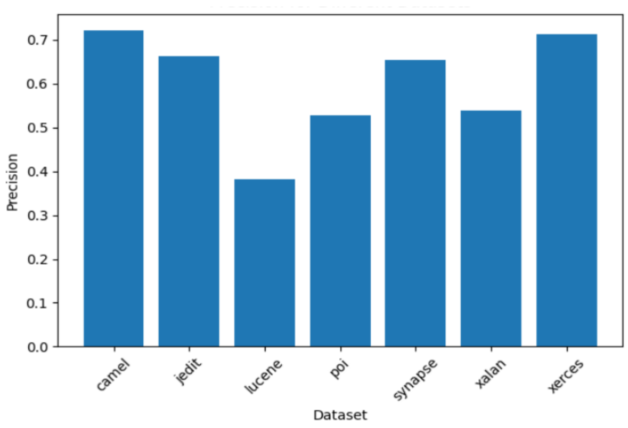

FIGURE 6. Precision metric of the proposed model

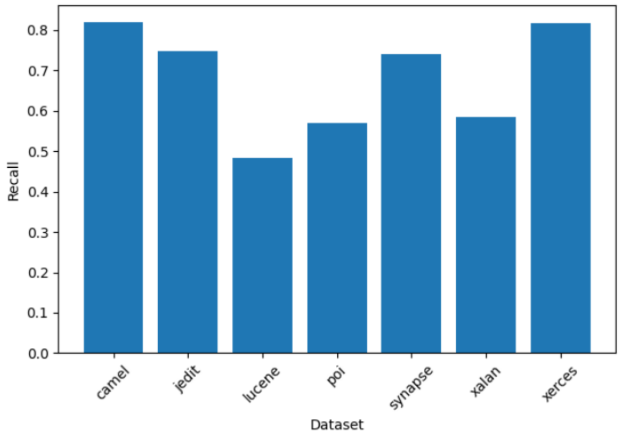

FIGURE 7. Recall metric of the proposed model

V. DISCUSSION

The current study proposed an advanced deep learning pipeline integrating CNN and Random Forest classifiers to address the limitations of traditional software defect prediction methods. The results demonstrated that the ensemble model outperformed the existing models in terms of accuracy and F1 score across multiple datasets. By utilizing CNN, semantic and structural features were captured from the software data, a notable improvement over hand-crafted features typically used in defect prediction. The CNN's ability to analyze grid-like data structures, such as source code, made it especially useful for identifying complex patterns within software projects. On the other hand, the Random Forest model provided robustness by handling the data imbalance effectively and offering strong prediction accuracy, especially in datasets with skewed distributions, such as those seen in Camel and Xerces projects.

The findings suggested that combining deep learning with traditional ML models can enhance the accuracy of software defect prediction. The ensemble approach, which aggregates the strengths of both CNN and Random Forest models through majority voting, provided better generalization and increased predictive performance. These results emphasize the potential of hybrid models in defect prediction and suggest new research avenues, such as exploring other deep learning architectures including LSTMs or graph neural networks.

A. CONCLUSION

This study introduced a novel deep learning approach for early SDP, combining the strengths of CNNs and Random Forest classifiers into a cohesive ensemble model. The methodology, validated across multiple publicly-available datasets, demonstrated superior performance over existing models in terms of accuracy, F1 score, precision, and recall.

The experimental results highlighted the efficacy of integrating deep learning techniques into the defect prediction domain, offering software developers a more reliable tool for identifying buggy code early in the development process. This, in turn, may lead towards more efficient allocation of testing resources and a reduction in post-release defects. Future research should focus on refining the model by incorporating additional neural network architectures and exploring its applicability in different software environments.

CONFLICT OF INTEREST

The authors of the manuscript have no financial or non-financial conflict of interest in the subject matter or materials discussed in this manuscript.

DATA AVAILABILITY STATEMENT

The dataset used is available from the corresponding author upon reasonable request.

FUNDING DETAILS

No funding has been received for this research.

REFERENCES

[1] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, “Defect prediction from static code features: Current results, limitations, and new approaches,” *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, Dec. 2010, doi: https://doi.org/10.1007/s10515-010-0079-7.

[2] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, “Dictionary learning based software defect prediction,” in *Proc. Int. Conf. Softw. Eng. (ICSE)*, May 2014, pp. 414–423, doi: https://doi.org/10.1145/2568225.2568230.

[3] S. Wang, T. Liu, and L. Tan, “Automatically learning semantic features for defect prediction,” in *Proc. Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 297–308, doi: https://doi.org/10.1145/2884781.2884804.

[4] T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 2–13, Jan. 2007, doi: https://doi.org/10.1109/TSE.2007.256941.

[5] T. J. McCabe, “A complexity measure,” *IEEE Trans. Softw. Eng.*, no. 4, pp. 308–320, Dec. 1976, doi: https://doi.org/10.1109/TSE.1976.233837.

[6] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object-oriented design,” *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi:

https://doi.org/10.1109/32.295895.

[7] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM models for speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280, doi: https://doi.org/10.1109/ICASSP.2012.6288864.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2012, pp. 1097–1105, doi: https://doi.org/10.1145/3065386.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: https://doi.org/10.1109/5.726791.

[11] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2015, pp. 649–657.

[12] D. Al-Fraihat and Y. Sharrab, "Predicting software defects using ensemble learning techniques," in *Proc. IEEE Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2022, pp. 1–10, doi: https://doi.org/10.1109/DSAA54385.2022.10032457.

[13] "The PROMISE repository of software engineering databases," in *Softw. Eng. Databases*, 2nd ed., vol. 3. NASA, Ed. Pennsylvania, USA: CiteSeerX, 2024, pp. 1–12.

[14] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *Proc. IEEE Int. Conf. Softw. Qual., Reliab. Secur. (QRS)*, Jul. 2017, pp. 318–328, doi:https://doi.org/10.1109/QRS.2017.42.

[15] Fei, "PROMISE-backup," GitHub repository. [Online]. Available: https://github.com/feiwww/PROMISE-backup. [Accessed: Aug. 3, 2024].

[16] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction," *IEEE Trans. Softw. Eng.*, vol. 34, no. 4, pp. 485–496, Jul. 2008, doi: https://doi.org/10.1109/TSE.2008.35.

[17] T. C. Sharma and M. Jain, "WEKA approach for comparative study of classification algorithms," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 2, no. 4, pp. 1925–1931, Apr. 2013.

[18] P. J. Kaur and Pallavi, "Data mining techniques for software defect prediction," *Int. J. Softw. Web Sci.*, vol. 3, no. 1, pp. 54–57, 2013.

[19] T. Wang, W. Li, H. Shi, and Z. Liu, "Software defect prediction based on classifiers ensemble," *J. Inf. Comput. Sci.*, vol. 8, no. 1, pp. 4241–4254, 2011.

[20] V. R. Basili *et al.*, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996, doi: https://doi.org/10.1109/32.544352.

[21] T. Gyimothy, R. Ferenc, and I. Siket,

"Empirical validation of object-oriented metrics on open source software," *IEEE Trans. Softw. Eng.*, vol. 31, no. 10, pp. 897–910, Oct. 2005, doi: https://doi.org/10.1109/TSE.2005.112.

[22] P. Singh and S. Verma, "Cross project software fault prediction at the design phase," *Int. J. Comput. Inf. Eng.*, vol. 9, no. 3, pp. 800–805, 2015.

[23] B. J. Park *et al.*, "Polynomial function-based neural networks for software defect detection," *Inf. Sci.*, vol. 229, pp. 40–57, Apr. 2013, doi: https://doi.org/10.1016/j.ins.2012.12.042.

[24] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, May 2008, doi: https://doi.org/10.1016/j.jss.2007.07.040.

[25] S. Shivaji *et al.*, "Reducing features to improve code change-based bug prediction," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 552–569, Apr. 2013, doi: https://doi.org/10.1109/TSE.2012.38.

[26] K. Dejaeger *et al.*, "Toward comprehensible software fault prediction models using Bayesian networks," *IEEE Trans. Softw. Eng.*, vol. 39, no. 2, pp. 237–257, Feb. 2013, doi: https://doi.org/10.1109/TSE.2012.24.

[27] S. S. Rathore and S. Kumar, "A decision tree logic-based recommendation system," *Computing*, vol. 99, no. 3, pp. 255–284, Mar. 2017, doi: https://doi.org/10.1007/s00607-016-0485-9.

[28] P. Singh and S. Verma, "Multi-classifier model for software fault prediction," *Int. Arab J. Inf. Technol.*, vol. 15, no. 5, pp. 912–919, Sep. 2018.

[29] P. Singh *et al.*, "Fuzzy rule-based approach for software fault prediction," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 47, no. 5, pp. 826–837, May 2017, doi: https://doi.org/10.1109/TSMC.2016.2566608.

[30] X. Yang *et al.*, "Deep learning for just-in-time defect prediction," in *Proc. IEEE Int. Conf. Softw. Qual., Reliab. Secur.*, Aug. 2015, pp. 17–26, doi: https://doi.org/10.1109/QRS.2015.14.

[31] T. Thaher and F. Khamayseh, "A novel machine learning approach for software defect prediction," in *Proc. Int. Conf. Comput. Intell. Data Eng.*, 2021, pp. 95–106.

[32] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in *Proc. Int. Symp. Softw. Reliab. Eng.*, Nov. 2009, pp. 99–108, doi: https://doi.org/10.1109/ISSRE.2009.33.

APPENDIX

Datasets: The study used seven publicly-available software defect datasets from the PROMISE repository. Key features extracted included code metrics (e.g., McCabe features) and process metrics (e.g., change history).

Data Preprocessing: Data was loaded and concatenated into a single data frame for unified training and evaluation. Separated features—comprising code metrics and process metrics—from the target labels (buggy/clean). Applied StandardScaler to numeric features and OneHotEncoder to categorical features, then split the data into training and testing sets.

Model Architecture

CNN Model: Extracted semantic and structural features using Conv1D, MaxPooling, Dense, and Dropout layers.

Random Forest Classifier: Built an ensemble of DTs; fine-tuned using GridSearchCV for optimal performance.

Ensemble Method: Combined predictions from both CNN and Random Forest models using majority voting.

Evaluation Metrics: Performance measured using accuracy, precision, recall, and F1 Score. F1 score emphasized for its utility in class-imbalanced datasets.

Tools and Libraries: Pandas and numpy for data manipulation. scikit-learn for ML, preprocessing, and hyperparameter tuning. Keras for building the CNN. matplotlib for generating performance plots. imblearn for handling class imbalances.

Results Visualization: Violin Plots visualized the distribution of performance metrics across datasets. Results are presented in tables, highlighting the superior performance of the ensemble method.

Ethical Considerations: Ensured responsible handling of open-sourced datasets with attention to privacy, transparency, and bias mitigation.