ENHANCING MOBILE APP QUALITY THROUGH DEFECTS PREDICTION USING CNN-BiLSTM MODEL

Muhammad, A.A.[1], Umar, K.[2]* and Agaie, A.I.[3]

[1] Department of Computer Science, Faculty of Computing, Bayero University Kano

[2]Department of Software Engineering, Faculty of Computing, Bayero University Kano

[3]Department of information and Media Studies, Faculty of Communication, Bayero University Kano

*Corresponding email: ukabir.se@buk.edu.ng

ABSTRACT

Mobile applications are widely used, but undetected defects can negatively impact user experience and business reputation. Ensuring defect detection before release is crucial for maintaining software overall quality and reliability. Traditional defect prediction models struggle to handle the complexity of mobile applications. These models rely on predefined rules and handcrafted features, which often results in poor generalization and high false positive rates. Existing deep learning models also face challenges in capturing both spatial and temporal relationships in mobile app code. This study developed a hybrid CNN-BiLSTM model to improve defect prediction accuracy. The proposed approach included feature selection, data preprocessing, and model training using a dataset from COMMIT GURU. The model was evaluated using ten Android application datasets and achieved an average accuracy of 95% and an AUC of 93%, significantly outperforming previous models, which attained only 69% accuracy with the same AUC. These results validated the effectiveness of the proposed CNN-BiLSTM hybrid model in accurately identifying software defects in mobile apps and highlighted its potential for improving software reliability. Future work should focus on improving the model's generalizability across diverse datasets and optimizing its computational efficiency to enhance scalability and real-time applicability.

Keywords: Software fault prediction; mobile application; Android applications; deep learning; machine learning

1 INTRODUCTION

The mobile application landscape has evolved rapidly, progressing from simple utilities such as contacts and calendars to sophisticated, feature-rich tools that significantly impact daily life [1, 2]. This evolution, driven by technological advancements, has introduced difficulties in mobile application development, particularly in testing and quality assurance. Unlike desktop and web applications, mobile apps must support various platforms, screen sizes, and interaction methods, making testing and maintenance inherently more difficult [3, 4]. As user expectations for reliability and usability continue to increase, even minor software defects can damage a developer's reputation. Traditional fault analysis techniques like static code inspection, black-box testing, and automation have been used to reduce defects [4], but they usually fall short in dynamic mobile environments. Predictive models that leverage historical software metrics and machine learning (ML) to anticipate defects have shown potential. However, most of these models have focused on desktop and web applications, achieving limited success in mobile contexts [2, 5]. Recent studies have sought to bridge this gap using ML and DL techniques. For example, [6] focused

on Android-specific predictions, and [2] investiged CNN, ANN, and LSTM models. Despite these advancements, difficulties such as overfitting and insufficient handling of sequential and long-term dependencies remain. This research proposes a hybrid CNN-BiLSTM model to enhance defect prediction accuracy in mobile applications. By integrating the spatial feature extraction capabilities of Convolutional Neural Networks (CNN) with the temporal modeling strength of Bidirectional Long Short-Term Memory (BiLSTM) networks, the model addresses the limitations of previous approaches. In summary, this study facilitates to mobile defect prediction by introducing a CNN-BiLSTM hybrid model, which was conducted and experimented with using Google Colab and evaluated on ten Android application datasets. The results showed that the proposed model achieved an average accuracy of 95% and an AUC of 93%, significantly outperforming other models. These findings validate the effectiveness of the CNN-BiLSTM approach in accurately identifying software faults in mobile applications, demonstrating improved accuracy and robustness. Overall, the study offers a viable solution for enhancing mobile software reliability. This study is organized into five key sections that collectively gives an in-depth exploration of deep learning-based defect prediction in Android mobile applications. Section 1 presents the Introduction while the Related Work section which is section 2 presents a critical review of existing literature on defect detection and prediction, synthesizing methodologies, findings, and limitations to identify research gaps. Section 3 present the Methodology which outlines the research design, including data collection, preprocessing, model development, and evaluation metrics. The Experimental Evaluation is presented in section 4, which details the implementation setup, performance comparisons, and analysis of results. Finally, section 5, the Conclusion and Future Work section consolidates the research outcomes, highlights key contributions, and offers recommendations for future research, emphasizing the study's implications and potential directions for further exploration.

2 LITERATURE REVIEW

This section presents a comprehensive review of literature on Software Fault Prediction (SFP), Mobile Application Defect Prediction (MADP), and numerous traditional and contemporary approaches, which includes Machine Learning (ML), Deep Learning (DL), Feature Selection (FS), and Search-Based Optimization. The goal of this review is to formulate a solid theoretical foundation and highlight existing advancements, thereby guiding the formulation and assessment of the proposed approach.

2.1 Software Fault Prediction (SFP)

Software Defect Prediction (SDP) represents a key area within software engineering that mainly focuses on detecting fault-prone components early in the development life cycle to enhance overall software quality. Previous research in this domain primarily made use of statistical techniques and classical machine learning models. Study [7] examined algorithms such as Artificial Neural Networks (ANNs), Bayesian Networks (BNs), and Rule Induction methods, finding out that integrating instance-based learning with consistency-driven feature selection substantially enhanced predictive performance.

2.2 Mobile Application Defect Prediction (MADP)

Unlike traditional Software Fault Prediction (SFP) approaches that mainly focus on desktop and web systems, mobile applications present distinct challenges such as platform diversity, sensor-driven interactions, frequent updates, and different execution contexts. According to [2], defect prediction in mobile environments plays a vital role in identifying and prioritizing high-risk components, while [8] reported its potential in minimizing maintenance costs and accelerating release cycles. Even so, the variability of mobile datasets and the scarcity of labeled data continue to prevent progress, specifically resulting from the widespread use of third-party APIs and the continuous versioning of mobile platforms.

2.3 Traditional Approaches

Conventional Software Defect Prediction (SDP) approaches initially utilized software metrics such as lines of code, cyclomatic complexity, cohesion, coupling, and code churn. Predictive modeling in these methods commonly used algorithms like logistic regression and decision trees. Study [9] showed that

correlation-based feature selection could improve model stability, though it may require extensive preprocessing techniques. Similarly, [10] presented aggregated change metrics that integrated both content and temporal information to enhance defect prediction performance; nonetheless, their applicability within different domains remained constrained. While these traditional models are valued for their simplicity and interpretability, they still struggle to capture the non-linear and intricate relationships characteristic of contemporary mobile applications.

2.4 Search-Based Approaches

Search-Based Software Engineering (SBSE) have become a robust methodology for optimizing numerous stages of the defect prediction process. Study [11] demonstrated the effectiveness of metaheuristic techniques like Genetic Algorithms (GA) and Simulated Annealing (SA) in automating test case generation, thereby enhancing code coverage and defect detection. Building on this, [12] experimented a graph-search model to handle concurrency issues in Business Process Execution Language (BPEL) applications, though its applicability was confined only in some specific domains. In [13], the Multi-Objective Defect Predictor (MODEP) was proposed, employing a genetic algorithm-based multi-objective approach to balance defect identification with inspection costs. Furthermore, [14] explored hybrid techniques, revealing that the GFS-LogitBoost model attained strong performance in early defect detection but needed substantial computational resources. Lastly, [15] combined SDP with search-based testing (EvoSuite), leveraging defect scores to prioritize high-risk code segments during the testing phase, which led to notable improvements in bug detection efficiency.

2.5 Machine Learning-Based Approaches

Machine learning techniques have shown impressive success in predicting defects in mobile applications, specifically in Just-In-Time (JIT) defect prediction scenarios. Study [16] introduced a JIT-SDP framework for Android applications that leveraged commit metadata and resource-specific features, achieving an excellent performance compared to conventional predictors. Building on this, [17] proposed a hybrid model combining Random Forest and Support Vector Machine (SVM), which achieved an impressive 98.79% accuracy, though its effectiveness was constrained by reliance on predefined bug characteristics. A comprehensive meta-analysis by [18] highlighted persistent challenges in JIT-SDP, which includes the need for dual data streams and limitations associated with projects containing few defects. Moreover, [19] underscored the significance of aligning datasets with appropriate algorithms, while [20] proposed unsupervised CNN-based approaches that achieved excellent results without labeled data but struggling from limited interpretability. Furthermore, [21] investigated reinforcement learning-based adaptive SDP models, which enable real-time adjustment to changing software environments, though transparency and explainability remained.

2.6 Deep Learning-Based Approaches

Deep learning has changed Software Fault Prediction (SFP) by facilitating the automatic extraction of sophisticated, hierarchical code representations. Study [2] showed that Convolutional Neural Networks (CNNs) surpassed other deep models in mobile SDP, which achieved an AUC of 0.933, though issues related to adaptability remained. However, [22], proposed a Deep Q-Network (DQN) model incorporating dynamic reward mechanisms that achieved a 27% improvement in accuracy but encountered complexities in designing effective reward functions. Similarly, [4] addressed class imbalance issues using a cost-sensitive Imbalanced Deep Learning (IDL) approach, while [23] explored the use of deep neural networks on APK files, proving the method's feasibility despite limited dataset diversity. Furthermore, [24] employed CNN-based deep learning for defect detection in building structures using smartphone imagery, underscoring the versatility and cross-domain applicability of deep learning techniques.

3 MATERIALS AND METHOD

This section presents the research workflow and outlines the proposed methodology employed in this study, covering all of the architecture of the model and subsequent performance evaluation.

3.1 The Research Workflow

This research was structured into four main phases as shown in fig. 1: literature review, methodology, implementation, and evaluation & results. Each phase contributed systematically to the development and validation of the hybrid deep learning model for predicting defects in Android mobile applications. In Phase One: Literature Review, a thorough review of existing literature was conducted. The review covered key areas including software defect prediction, deep learning-based defect prediction, mobile application defect prediction, and feature selection techniques. This phase provided a strong theoretical foundation for the study and identified gaps in current approaches that the proposed model aimed to address. Phase Two: Methodology outlined the architecture of the proposed model, which combined a hybridized Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (BiLSTM) networks. The model is particularly designed to predict defects in Android mobile applications. The CNN layer is employed to extract relevant features from commit-level data, while BiLSTM is meant to capture contextual and sequential dependencies. By leveraging the strengths of both networks, the model is assumed to enhance prediction accuracy and address the issue of functional failures in mobile applications. In Phase Three: Implementation, the proposed CNN-BiLSTM model is created and implemented using the Google Colab, which is a cloud-based environment. Python was used as the programming language, and the implementation involved building the model architecture, preprocessing the data, training the model, and lastly preparing it for evaluation. This phase provided the groundwork for empirical validation, and further information of the implementation were also presented in the results. Phase Four: Evaluation and Results only focused on assessing the performance of the developed model. The experimental setup was configured within the Google Colab environment, using a Windows operating system. The Keras deep learning framework was used alongside supporting Python libraries which includes, Pandas, NumPy, and Scikit-learn for data preprocessing, and Matplotlib for data visualization. The evaluation was conducted using ten open-source Android application datasets gotten from previous research (2) and accessed via the COMMIT GURU platform. These datasets contained 31 features and 34,042 instances of varying sizes. The model's performance was evaluated using standard metrics including accuracy, precision, recall, F1-score and Area Under Curve (AUC). Furthermore, a cross-validation strategy was employed, using 80% of the dataset for training and 20% for testing, so as to ensure a robust and reliable assessment of the model's effectiveness.

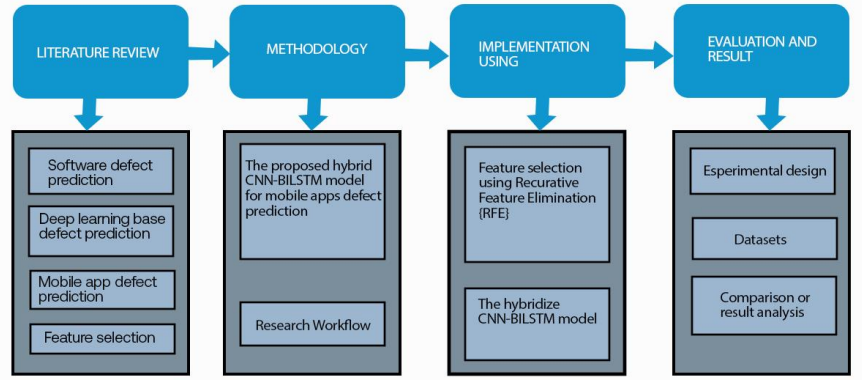

Figure 1. Research workflow

3.2 The Proposed CNN-BiLSTM Based Defect Prediction Model

This section provides a detailed description of the methodology adopted for defect prediction in Android mobile applications using a hybrid CNN-BiLSTM model. The model, illustrated in Figure 2, is structured into five stages which includes: dataset acquisition, data preprocessing, feature selection and class

balancing, hybrid model development, and lastly performance evaluation. i. Stage 1 involved the acquisition of a dataset which comprised commit-level information from Android mobile applications. Each record contained software metrics and a corresponding binary label that indicates whether the commit introduced a defect or not. Due to the imbalance nature of the dataset where non-defective commits significantly outnumbered defective ones, additional data processing was required to mitigate bias and improve model performance. ii. Stage 2 focused on data preprocessing to prepare it suitable for learning. This stage starts with data cleaning, during which irrelevant and redundant features such as repository_id, author_email, and issue_id was dropped. Missing values in numerical fields were imputed using the mean strategy, while categorical fields such as fix were filled with placeholder values. All categorical and boolean variables were encoded into numerical format using label encoding. Numerical features were standardized using z-score normalization to ensure consistent feature scaling. Textual data within the commit_message field were transformed using Term Frequency-Inverse Document Frequency (TF-IDF), with the top 100 most informative terms retained. Lastly, the target variable was binarized such that any non-zero-defect count was treated as a defective class. iii. Stage 3 addresses feature selection and class imbalance. Recursive Feature Elimination (RFE) was used to reduce feature dimensionality and retain only the most predictive attributes, thereby reducing computational overhead. To balance the dataset, Synthetic Minority Over-sampling Technique (SMOTE) was employed to generate synthetic samples for the minority class (defective commits), thereby creating a more balanced training set and to improve model generalization. iv. Stage 4 involved the implementation of the hybrid CNN-BiLSTM model. The preprocessed and balanced dataset was passed through a Convolutional Neural Network (CNN) layer, which extracted spatial features and local patterns, specifically from the transformed commit messages. The output from the CNN layer was then fed into the Bidirectional Long Short-Term Memory (BiLSTM) layer, which captured sequential dependencies in both forward and backward directions. This combination allowed the model to learn both contextual and temporal relationships across software metrics and commit histories. A dense layer is then followed, integrating the BiLSTM outputs to capture high-level abstract patterns. The final output was produced using a binary output layer with a sigmoid activation function, which generated probabilities for defect prediction and subsequently classified commits as defective or not. v. Stage 5 concluded the process with performance evaluation. The predictive capability of the proposed model was assessed using standard evaluation metrices including accuracy, precision, recall, F1-score and AUC.

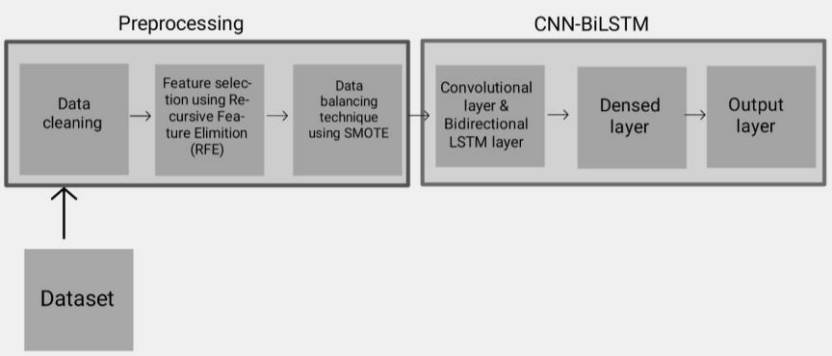

Figure 2. Hybridized CNN-BiLSTM model architecture.

4 RESULTS AND DISCUSSION

This section is devoted to the presentation of the experimental results.

4.1 CNN-BiLSTM model results with imbalanced data

Table 1 presents the evaluation results of the CNN-BiLSTM model on ten imbalanced Android mobile application datasets which are Alfresco, Any Soft Keyboard, Chat Secure Android, Flutter, PageTuner, Afwell, Android_Wallpaper Designer, Apg, Atmosphere, and Facebook-Android-sdk using five standard classification metrics: Accuracy, Precision, Recall, F1-Score, and AUC (Area Under Curve). Each

dataset is listed with its corresponding values for these metrics, the final row provides the overall average values across all datasets, giving a concise summary of the collective results.

Table 1. CNN-BiLSTM model performance with imbalanced data

S/N	Datasets	Accuracy	Precision	Recall	F1-Score	AUC
1	Alfresco	0.4731	0.2348	0.2100	0.3803	0.2857
2	AnySoftKeyboard	0.5333	0.2731	0.4106	0.4146	0.4146
3	ChatSecureAndroid	0.4089	0.3307	0.3307	0.4941	0.3737
4	Flutter	0.1980	0.0065	0.2237	0.0130	0.2837
5	PageTuner	0.5714	0.4516	0.4233	0.6087	0.3519
6	Afwell	0.5507	0.4690	0.2415	0.6239	0.4142
7	Android_WallpaperDesigner	0.3208	0.2030	0.3910	0.3333	0.3578
8	Apg	0.5458	0.3635	0.1871	0.5031	0.2283
9	Atmosphere	0.5466	0.3624	0.2647	0.4587	0.5683
10	Facebook-Android-sdk	0.4280	0.2029	0.3325	0.3286	0.3032
	Average =	0.4576	0.2897	0.3811	0.4158	0.5125

4.2 CNN-BiLSTM model results with balanced data.

Table 2 below shows the results of the proposed CNN-BiLSTM model with balanced datasets. It presents the evaluation outcomes of ten Android mobile application datasets using five classification metrics: Accuracy, Precision, Recall, F1-Score, and AUC. Each dataset is reported with its respective values across the metrics, the final row provides the overall average results. Compared with the results obtained from the imbalanced datasets, these balanced dataset results show a significant improvement across all metrics, highlighting the effectiveness of addressing class imbalance in enhancing the reliability and consistency of prediction outcomes.

Table 2. CNN-BiLSTM model performance with balanced data.

S/N	Datasets	Accuracy	Precision	Recall	F1-Score	AUC
1	Alfresco	0.7904	0.3947	0.5556	0.4615	0.6956
2	AnySoftKeyboard	0.9977	1.0000	0.9880	0.9940	0.9940
3	ChatSecureAndroid	0.9880	1.0000	0.9593	0.9792	0.9797
4	Flutter	0.9999	1.0000	0.9737	0.9867	0.9868
5	PageTuner	0.8571	0.9091	0.6667	0.7692	0.8148
6	Afwell	0.9781	0.9929	0.9521	0.9720	0.9737
7	Android_WallpaperDesigner	0.9937	1.0000	0.9655	0.9825	0.9828
8	Apg	0.9908	0.9798	0.9878	0.9838	0.9899
9	Atmosphere	0.9951	0.9946	0.9893	0.9919	0.9934
10	Facebook-Android-sdk	0.9797	1.0000	0.8750	0.9333	0.9375
	Average =	0.9571	0.9271	0.8913	0.9054	0.9348

The results from the proposed model in this study indicates strong performance across most datasets, with a few exceptions. With the proposed model having a very good performance on Any Soft Keyboard, Chat Secure Android, and Flutter in terms of accuracy, precision, recall, F1 score, and AUC (Area Under Curve). The proposed model in this study is effective at correctly identifying defects (positive instances) and minimizing false positives. For instance, Any Soft Key board has perfect precision, meaning that when it predicts a defect, it's almost always right. However, the proposed model did not perform very

well on Alfresco, with low accuracy, precision, and recall, the proposed model in this study struggles to accurately identify defects or avoid incorrectly labeling non-defective parts of the app as faulty. In the context of mobile app defect prediction, this means that the proposed model in this study is less reliable at identifying which areas of the app actually need attention, which can either leads to missed defects or false alarms. Overall, the proposed model in this study performs well on average, with an accuracy of 0.9571 and an AUC of 0.9348, indicating good ability to differentiate between defective and non-defective parts of the apps. The average F1 score of 0.9054 indicates a solid balance between identifying defects and avoiding false positives across the datasets. While the proposed model in this study performs well on most datasets, the Alfresco dataset is an outlier that impacts the overall results. The line graph and bar chart of the datasets are displayed in figure 3 and 4. i. The line graph effectively showcases the consistency of the proposed model's performance on most datasets while clearly indicating which datasets pose greater prediction challenges.

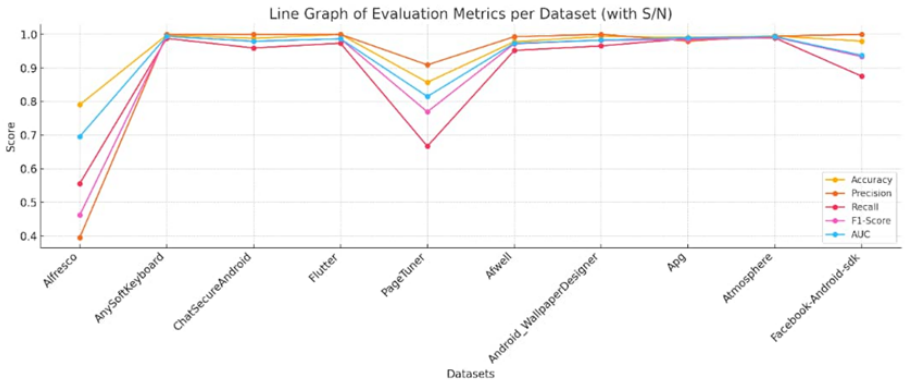

Figure 3. Line Graph showcasing Evaluation metrices per Datasets.

ii. The bar chart complements the line graph by providing a more direct visual comparison of metric values within each dataset, effectively emphasizing both the strengths and areas for improvements in the proposed model's performance.

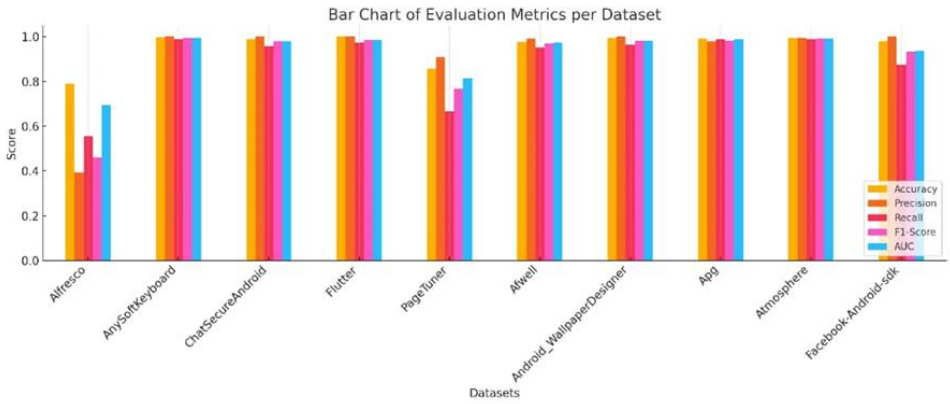

Figure 4. Bar Chat

4.3 Results Comparison

In this work, we have compared our proposed CNN-BiLSTM model with three deep learning models from the work of [2], which are CNN, ANN and LSTM. Our proposed CNN-BiLSTM model is the best performer, excelling in accuracy, precision, recall, F1-score, and AUC, making it ideal for mobile app defect prediction due to its combination of convolutional layers and BiLSTM layers. For the base work that have three classifiers, i.e work of [2], which are CNN, ANN and LSTM, the CNN ranks second with strong AUC, indicating good class distinction, but its accuracy and recall are lower, affecting overall performance

of the model. ANN and LSTM are moderate performers, with LSTM slightly better in recall, but neither reaches the effectiveness of our CNN-BiLSTM model or the CNN model from the work of [2]. However, our CNN-BiLSTM performance without data balancing technique shows a very poor unsatisfactory performance, which significantly shows that the data balancing technique is undeniably crucial in obtaining satisfactory results.

Table 3: CNN-BiLSTM model results comparison.

Algorithms	Accuracy	Precision	Recall	F1-Score	AUC
CNN-BiLSTM (Proposed model on balanced data)	0.95	0.92	0.89	0.90	0.93
CNN-BILSTM (On imbalanced data)	0.45	0.28	0.38	0.41	0.51
CNN	0.62	0.76	0.71	0.73	0.913
ANN	0.70	0.75	0.76	0.75	0.91
LSTM	0.70	0.75	0.77	0.74	0.91

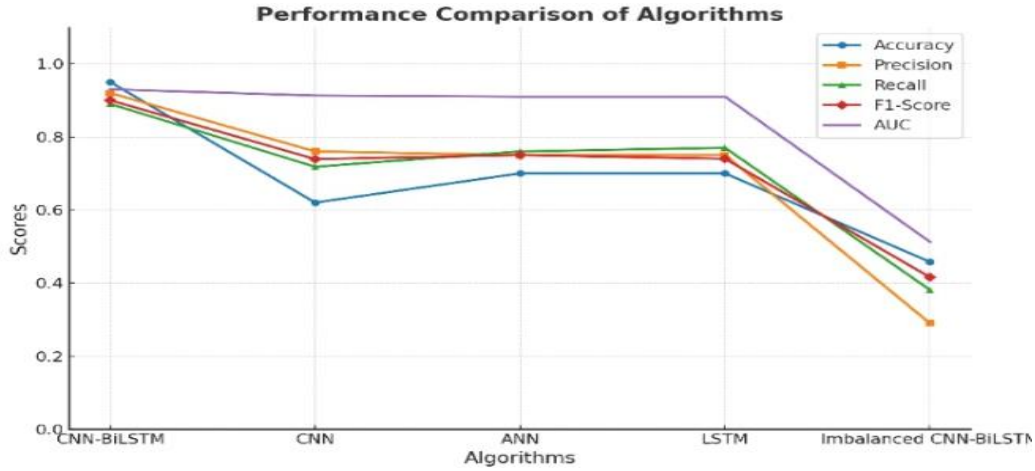

Figure 5. Line graph showcasing performance of the algorithms.

5 CONCLUSIONS

The proposed study achieved significant improvements over existing works, consistently outperforming previous methods across all evaluated metrics and datasets. These results validate the effectiveness of the proposed approach in meeting the research goals and enhancing prediction accuracy. Future studies should aim to improve the model's generalizability by testing it on a larger variety of datasets with diverse properties. Additionally, refining the model's architecture and employing metaheuristic-based hyperparameter optimization can help boost computational efficiency, scalability, and suitability for large-scale or real-time applications.

REFERENCES

1. Chen, Y., Zheng, B., Zhang, Z., Wang, Q., Shen, C. and Zhang, Q. (2020). Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. *ACM Comput. Surv. CSUR*, vol. 53, no. 4, pp. 1-37.

2. Jorayeva, M., Akbulut, A., Catal, C. and Mishra, A. (2022). Deep learning-based defect prediction for mobile applications. *Sensors*, vol. 22, no. 13, pp. 4734.

3. Kaur, A., Kaur, K. and Kaur, H. (2015). An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. *Presented at the 2015 4th International*

Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), IEEE, pp. 1-6.

4. Kaur, A., Kaur, K. and Kaur, H. (2015). An investigation of the accuracy of code and process metrics for defect prediction of mobile applications. *Presented at the 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), IEEE*, pp. 1-6.
5. Yahya, A.E., Gharbi, A., Yafooz, W.M. and Al-Dhaqm, A. (2023). A novel hybrid deep learning model for detecting and classifying non-functional requirements of mobile apps issues. *Electronics*, vol. 12, no. 5, pp. 12-58.
6. Zhao, K., Xu, Z., Zhang, T., Tang, Y. and Yan, M. (2021). Simplified deep forest model based just-in-time defect prediction for android mobile apps. *IEEE Trans*. Reliab., vol. 70, no. 2, pp. 848-859.
7. Challagulla, V.U.B., Bastani, F.B., Yen, I.L. and Paul, R.A. (2008). Empirical assessment of machine learning based software defect prediction techniques. *Int. J. Artif. Intell. Tools*, vol. 17, no. 2, pp. 389-400.
8. Akbulut, F.P., Perros, H.G. and Shahzad, M. (2020). Bimodal affect recognition based on autoregressive hidden Markov models from physiological signals. *Comput. Methods Programs Biomed.*, vol. 195, pp. 105571.
9. Arisholm, E., Briand, L.C. and Johannessen, E.B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Jour. Syst. Softw*., vol. 83, no. 1, pp. 2-17.
10. Šikić, L., Afrić, P., Kurdija, A.S. and ŠIlić, M. (2021). Improving software defect prediction by aggregated change metrics. *IEEE Access*, vol. 9, pp. 19391-19411.
11. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J. and McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. *Jour. Syst. Softw*., vol. 86, no. 8, pp. 1978-2001.
12. Morasca, S. (2006). On the Assessment of the Mean Failure Frequency of Software in Late Testing. *Presented at the 2006 International Conference on Software Engineering Advances (ICSEA'06), IEEE*, 2006, pp. 15-15.
13. Canfora, G., Lucia, A.D., Penta, M.D., Oliveto, R., Panichella, A. and Panichella, S. (2015). Defect prediction as a multiobjective optimization problem. *Softw. Test. Verification Reliab*., vol. 25, no. 4, pp. 426-459.
14. Rhmann, W. (2018). Application of hybrid search-based algorithms for software defect prediction. *Int. J. Mod. Educ. Comput. Sci.*, vol. 12, no. 4, pp. 1-51.
15. Perera, A., Aleti, A., Böhme, M. and Turhan, B. (2020). Defect prediction guided search-based software testing. *Presented at the Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pp. 448-460.
16. Zhao, K., Xu, Z., Zhang, T., Tang, Y. and Yan, M. (2021). Simplified deep forest model based just-in-time defect prediction for android mobile apps. *IEEE Trans. Reliab*., vol. 70, no. 2, pp. 848-859.
17. Jadhav, V., Devale, P., Jadhav, R., Bidwe, R., Mane, M. and Pawar, P. (2024). Machine Learning and Just-in-Time Strategies for Effective Bug Tracking in Software Development. *Int. J. Intell. Syst. Appl. Eng*., vol. 12, pp. 749-758.
18. Zhao, Y., Damevski, K. and Chen, H. (2023). A systematic survey of just-in-time software defect prediction. *ACM Comput. Surv*., vol. 55, no. 10, pp. 1-35.

19. Mehmood, I., Shahid, S., Hussain, H., Khan, I.B., Ahmad, S., Rahman, S., Ullah, N. and Huda, S. (2023). A novel approach to improve software defect prediction accuracy using machine learning. *IEEE Access*, vol. 11, pp. 63579-63597.

20. Zhong, B., Xing, X., Love, P., Wang, X. and Luo, H. (2019). Convolutional neural network: Deep learning-based classification of building quality problems. *Adv. Eng. Inform.*, vol. 40, pp. 46-57.

21. Ardimento, P., Aversano, L., Bernardi, M.L., Cimitile, M. and Iammarino, M. (2022). Just-in-time software defect prediction using deep temporal convolutional networks. *Neural Comput. Appl*., vol. 34, no. 5, pp. 3981-4001.

22. Ismail, A.M., Ab Hamid, S.H., Sani, A.A. and Daud, N.N.M. (2024). Toward reduction in false positives just-in-time software defect prediction using deep reinforcement learning. *IEEE Access*, vol. 12, pp. 47568-47580.

23. Catal, C., Giray, G. and Tekinerdogan, B. (2022). Applications of deep learning for mobile malware detection: A systematic literature review. *Neural Comput. Appl*., vol. 34, no. 2, pp. 1007-1032.

24. Perez, H. and Tah, J.H. (2021). Deep learning smartphone application for real-time detection of defects in buildings. *Struct. Control Health Monit*., vol. 28, no. 7, pp. e2751.