International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online) Volume 15, Issue 7, July 2026)

Review of Software Defect Prediction Using Deep Learning Technique

[1]Shama Parveen, [2]Dr. Navneet Kaur
[1]*M.Tech Scholar, Department of Computer Science and Engineering, Sagar Institute of Research & Technology, Bhopal, India*
[2]*Professor, Department of Computer Science and Engineering, Sagar Institute of Research & Technology, Bhopal, India*

Abstract- Software defect prediction is an important area of software engineering that aims to identify defect-prone software components before they cause failures during testing or deployment. This review focuses on the use of deep learning techniques for software defect prediction, examining how models such as Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and hybrid deep learning approaches are applied to software metrics and source-code data. The review analyzes commonly used datasets, input features, model architectures, evaluation measures, and prediction performance reported in existing studies. Deep learning methods can automatically learn complex patterns and relationships from large-scale software data, reducing the dependence on manually designed features. However, challenges such as imbalanced datasets, limited training data, model interpretability, computational requirements, and variation among software projects remain important concerns.

Index Terms- Software defects, Machine, Deep, Prediction.

I. INTRODUCTION

Software has become an essential component of modern business, communication, healthcare, transportation, finance, and other critical systems. As software systems increase in size and complexity, the possibility of defects during development also increases, making software quality assurance a major concern. Software Defect Prediction (SDP) is a software engineering activity that aims to identify software modules, classes, files, or other components that are likely to contain defects before they cause failures or require extensive testing. By identifying defect-prone components at an early stage, development teams can prioritize testing and maintenance resources more effectively and reduce the overall effort required for software quality assurance [1]. Early research in SDP has explored different statistical, machine learning, data mining, and computational intelligence techniques for predicting defective software components using measurable software characteristics [2].

A major foundation of SDP is the use of software metrics, which provide quantitative information about software characteristics such as code size, complexity, coupling, cohesion, inheritance, and other structural properties. Static code attributes have been widely investigated for constructing defect prediction models, with studies showing that these attributes can provide useful information for distinguishing defective and non-defective software components [3]. Classification-based approaches have also become an important direction in SDP because they can categorize software components according to their likelihood of being defective. Comparative studies involving multiple classifiers and public datasets have demonstrated the usefulness of metric-based classification while also emphasizing the importance of appropriate evaluation procedures and datasets [4]. In addition, dependency relationships among software components can provide valuable information because highly connected or central components may have greater exposure to defects and can therefore be considered during prediction [5].

The availability of public software repositories and datasets has further supported the development and comparison of SDP techniques. Public datasets allow researchers to conduct repeatable experiments and evaluate different prediction models under similar conditions. However, software defect datasets frequently exhibit class imbalance, where the number of non-defective components is considerably larger than the number of defective components. Such imbalance can affect model training and may result in misleading performance when conventional accuracy is used as the primary evaluation measure [6]. Consequently, researchers have investigated cost-sensitive learning, ensemble methods, sampling strategies, and other techniques to improve prediction performance under imbalanced conditions [7]. The selection of appropriate performance measures, including precision, recall, F1-score, and area under the ROC curve, is therefore important for obtaining a reliable assessment of defect prediction models [8].

Another important direction is early software defect prediction, in which defect-prone components are identified during the requirement or design stages rather than waiting until coding or testing.

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online) Volume 15, Issue 7, July 2026)

Predicting defects earlier can support better project planning, testing prioritization, and resource allocation because potential quality problems can be addressed before they become expensive to correct [9]. However, prediction models developed for one software project may not always perform effectively when applied to another project because software projects can differ in development processes, domains, coding practices, and data distributions. Cross-project studies have therefore highlighted the difficulty of transferring defect prediction models between projects and the importance of understanding the characteristics of both source and target projects [10].

With the development of artificial intelligence, machine learning techniques have increasingly been applied to SDP. Systematic reviews have reported the use of decision trees, support vector machines, Bayesian methods, neural networks, ensemble learning, and other machine learning approaches for identifying defect-prone software components [11]. More recently, deep learning has emerged as an important approach because it can learn complex representations from software metrics, source code, and other software artifacts with less dependence on manually engineered features. Deep learning models such as Deep Neural Networks, Convolutional Neural Networks, Long Short-Term Memory networks, autoencoders, and tree-based neural architectures have been investigated for software defect prediction. Some approaches specifically attempt to capture the syntactic and semantic characteristics of source code rather than relying only on conventional complexity metrics [12]. Overall, software defect prediction has evolved from traditional metric-based and statistical techniques toward advanced machine learning and deep learning approaches, creating opportunities for more accurate, scalable, and automated identification of defect-prone software components.

II. REVIEW OF LITERATURE

He et al. [1] presented an explainable neural additive model for software defect prediction. The study focused on improving the interpretability of deep learning-based defect prediction by identifying how individual software features contribute to the prediction outcome. The approach provides a more transparent alternative to complex black-box models while maintaining predictive capability. The work highlights the importance of explainability for practical adoption of AI-based software defect prediction.

Mythili et al. [2] proposed an optimized Bi-directional Long Short-Term Memory (Bi-LSTM) model for software defect prediction. The model uses information from both forward and backward sequences to learn complex relationships within software data. Optimization was incorporated to improve the prediction capability of the deep learning architecture. The study demonstrates the potential of Bi-LSTM for identifying defect-prone software components.

Fu et al. [3] reviewed the role of artificial intelligence in DevSecOps and discussed emerging opportunities for AI-based software development and security. The study examined how AI techniques can support different stages of the software lifecycle, including testing, security analysis, and quality assurance. It emphasized automation and intelligent decision-making as important benefits of AI integration. The work provides a broader perspective on the application of AI for improving software development practices.

Table 1: Performance [1]

Project	QT	JDT	OpenStack	Platform	Gerrit	Go
AUC	0.7600	0.7575	0.7645	0.7789	0.8157	0.7425
Precision	0.6836	0.6823	0.6876	0.6995	0.7399	0.6883
Recall	0.7079	0.6659	0.7050	0.7154	0.7386	0.6544
F1-score	0.6955	0.6740	0.6962	0.7073	0.7393	0.6709

Mehmood et al. [4] developed a machine learning-based approach to improve software defect prediction accuracy. The study investigated different software characteristics and machine learning techniques for distinguishing defective and non-defective modules. The proposed methodology aimed to improve prediction performance compared with conventional approaches. The results demonstrate the usefulness of machine learning for supporting early identification of software defects.

Muthurajkumar and Kumar [5] introduced a hybrid deep learning architecture combining Swin Transformer and CNN concepts for image-based disease prediction. Although the study focused on cotton disease rather than software defects, it demonstrates the ability of hybrid deep learning architectures to extract complementary patterns from complex data. The findings indicate that combining different neural architectures can improve classification performance. This concept can also provide motivation for developing hybrid models for software defect prediction.

Kaladevi et al. [6] proposed an LSTM-based approach for crime analysis using Twitter time-series data. The model was designed to learn temporal patterns from sequential data and use them for prediction. The study demonstrates the effectiveness of LSTM networks in handling time-dependent information and capturing relationships across sequential observations. Such temporal learning capability can be relevant to software defect prediction when historical project or software evolution data are considered.

Kabila et al. [7] developed a hybrid LSTM-RNN model combined with the Lion optimization algorithm for proactive healthcare data management. The integration of recurrent neural networks enabled the model to capture sequential relationships, while optimization was used to improve the learning process. The study demonstrates

International Journal of Recent Development in Engineering and Technology
Website: www.ijrdet.com (ISSN 2347 - 6435 (Online) Volume 15, Issue 7, July 2026)

the usefulness of combining deep learning and optimization techniques for complex prediction tasks. This approach provides a potential direction for optimizing deep learning models used in software defect prediction.

Muthusankar et al. [8] proposed a Bidirectional Recurrent Neural Network (BiDRN) for sentiment analysis. The model processes sequential information in both directions to obtain richer contextual representations from the input data. The study demonstrates how bidirectional recurrent architectures can improve the understanding of relationships within sequential information. The same learning principle can be useful in software defect prediction where relationships among software metrics or sequential development information need to be captured.

Kumar et al. [9] presented a deep CNN-based approach combined with handcrafted features for sign language recognition. The study demonstrated that integrating automatically learned deep features with manually designed features can provide more informative representations for classification. This hybrid feature-learning strategy can be valuable for applications where individual feature types capture different characteristics. A similar combination of learned and traditional software metrics may improve defect prediction performance.

Gnanabaskaran et al. [10] proposed a graph neural network and support vector machine-based approach for drug-drug interaction prediction. The graph neural network was used to learn relationships among interconnected entities, while SVM supported the final classification process. The study highlights the importance of modeling relationships between interconnected data elements. This concept is relevant to software defect prediction because software components are also connected through dependencies, calls, inheritance, and other relationships.

Azzeh et al. [11] examined the performance of kernel methods for software defect prediction using Support Vector Machines. The study investigated how different kernel-based approaches can influence the classification of defective and non-defective software components. The findings provide insights into the effectiveness of SVM-based methods for handling software defect prediction problems. The work also establishes a useful comparison point for evaluating newer deep learning approaches against traditional machine learning techniques.

Praveen et al. [12] proposed a fusion architecture combining Convolutional Neural Networks and Capsule Networks for classification. The approach integrates the feature extraction capability of CNNs with the representation advantages of capsule-based learning. The study demonstrates that combining different deep learning architectures can improve classification effectiveness. This provides a useful basis for exploring hybrid deep learning architectures for software defect prediction, particularly when software data contain complex and diverse patterns.

Table 2: Summary of literature review

Sr. No	Author detail	Work	Outcome	Research Gap
1	He et al. (2025)	Understanding Software Defect Prediction Through eXplainable Neural Additive Models	Improved defect prediction interpretability.	Limited validation on diverse projects.
2	Mythili et al. (2025)	Software Defect Prediction System Using Optimized Bi-Directional LSTM	Bi-LSTM improved defect classification.	Needs testing on more datasets.
3	Fu et al. (2024)	AI for DevSecOps: A Landscape and Future Opportunities	Highlighted AI applications in DevSecOps.	Limited practical defect prediction solutions.
4	Mehmood et al. (2023)	A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning	Improved software defect prediction accuracy.	Limited deep feature learning.
5	Muthurajkumar et al. (2023)	SwinCNN: A Hybrid Deep Learning Architecture for Accurate Cotton Disease Prediction	Hybrid deep learning improved classification.	Not designed for software defects.

6	Kaladevi et al. (2024)	Tweets to Predict: LSTM Model for Crime Analysis Using Twitter Time Series Data	LSTM captured temporal patterns effectively.	Not evaluated for software defect data.
7	Kabila et al. (2024)	Hybrid LSTM-RNN and Lion Optimization Algorithm for IoT-Based Proactive Healthcare Data Management	Hybrid LSTM-RNN improved prediction.	Requires application to SDP.
8	Muthusankar et al. (2023)	BiDRN: A Method of Bidirectional Recurrent Neural Network for Sentiment Analysis	Bi-RNN improved sequential data classification.	Not applied to software defects.
9	Kumar et al. (2024)	Enhancing Sign Language Recognition Through Deep CNN and Handcrafted Features	Feature fusion improved classification.	Not tested with software metrics.
10	Gnanabaskaran et al. (2024)	Enhanced Drug-Drug Interaction Prediction with Graph Neural Networks and SVM	GNN-SVM effectively modeled relationships.	Software dependency graphs need exploration.
11	Azzeh et al. (2023)	Examining the Performance of Kernel Methods for Software Defect Prediction Based on Support Vector Machine	SVM kernel methods performed effectively.	Needs comparison with modern deep learning.
12	Praveen et al. (2023)	Convolutional Neural Network and Capsule Network Fusion for Effective Attrition Classification	CNN-Capsule fusion improved classification.	Not evaluated for software defect prediction.

III. CHALLENGES

Software Defect Prediction (SDP) using deep learning faces several challenges related to data quality, model complexity, feature representation, class imbalance, interpretability, and generalization. Although deep learning models can automatically learn complex patterns from software data, their effectiveness depends strongly on the availability of sufficient and reliable training data. Differences between software projects also make it difficult to develop a single model that performs consistently across different datasets. The major challenges are summarized below:

1. **Class Imbalance**: Software datasets generally contain more non-defective modules than defective modules. This imbalance can cause deep learning models to favor the majority class and miss actual defective components.

2. **Limited Training Data**: Deep learning models require large amounts of training data to learn effectively. Many software projects have limited labeled defect data, which can reduce model performance and increase the risk of overfitting.

3. **Feature Representation**: Software defects depend on complex relationships among code metrics, source-code structures, dependencies, and development history. Selecting or learning suitable representations remains challenging.

4. **Model Interpretability**: Deep learning models often work as black boxes, making it difficult to understand why a particular software component is predicted as defective. This limits trust and practical use by software developers.

5. **Cross-Project Generalization**: A model trained on one software project may not perform well on another project because of differences in programming languages, coding practices, project size, and development environments.

6. **High Computational Cost**: Training complex deep learning architectures requires considerable computational resources, processing time, and memory, particularly when large source-code datasets are used.

7. **Overfitting Problem**: Deep learning models can memorize patterns from small or project-specific datasets instead of learning general defect characteristics. This can lead to high training performance but poor performance on unseen data.

8. **Evaluation and Reliability**: Accuracy alone may not provide a reliable assessment because of class imbalance. Therefore, measures such as precision, recall, F1-score, and AUC should also be considered to properly evaluate defect prediction performance.

IV. CONCLUSION

Software defect prediction has evolved from traditional statistical and machine learning methods toward advanced deep learning approaches that can automatically learn complex patterns from software data. The reviewed studies show that models such as Bi-LSTM, CNN, RNN, GNN, and hybrid architectures can improve defect classification and software quality analysis. However, challenges such as class imbalance, limited datasets, overfitting, model interpretability, computational cost, and poor cross-project generalization still remain. Therefore, future research should focus on developing accurate, explainable, lightweight, and generalized deep learning models that can reliably predict software defects across different projects and development environments.

REFERENCES

1. R. He, Y. Li and C. Sun, "Understanding Software Defect Prediction Through eXplainable Neural Additive Models," in *IEEE Access*, vol. 13, pp. 82860-82873, 2025, doi: 10.1109/ACCESS.2025.3567140.
2. J. Mythili, T. Suganraj, V. Suvetha, M. Thamaraikannan and S. M. Polisetty, "Software Defect Prediction System Using Optimized Bi-Directional LSTM," 2025 International Conference on Advanced Computing Technologies (ICoACT), Sivalasi, India, 2025, pp. 01-06, doi: 10.1109/ICoACT63339.2025.11004992.
3. M. Fu, J. Pasuksmit, and C. Tantithamthavorn, "Ai for devsecops: A landscape and future opportunities," arXiv preprint arXiv: 2404.04839, 2024.
4. I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, "A novel approach to improve software defect prediction accuracy using machine learning," IEEE Access, vol. 11, pp. 63 579–63 597, 2023.
5. S. Muthurajkumar and G. K. Kumar, "Swincnn: A hybrid deep learning architecture for accurate cotton disease prediction," in 2023 12th Inter-national Conference on Advanced Computing (ICoAC). IEEE, 2023, pp. 1–7.
6. P. Kaladevi, M. Gopalraj, K. Harish, A. Guna, and R. Praveen, "Tweets to predict: Lstm model for crime analysis using twitter time series data," in 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS), vol. 1. IEEE, 2024, pp. 1–6.
7. R. Kabila, S. S. Balaji, V. Vikraam, S. R. Kumar, and R. Praveen, "Hybrid lstm-rnn and lion optimization algorithm for iot-based proactive healthcare data management," in 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS), vol. 1. IEEE, 2024, pp. 1–7.
8. D. D. Muthusankar, D. P. Kaladevi, D. V. Sadasivam, and R. Praveen, "Bidrn: A method of bidirectional recurrent neural network for sentiment analysis," arXiv preprint arXiv: 2311.07296, 2023.
9. K. D. Kumar, K. Ragul, G. P. P. Kumar, and G. K. Kumar, "Enhancing sign language recognition through deep cnn and handcrafted features," in 2024 2nd International Conference on Networking and Communications (ICNWC). IEEE, 2024, pp. 1–6.
10. A. Gnanabaskaran, K. T. Bharathi, S. Nandakumar, B. Sanjay, and R. Praveen, "Enhanced drug-drug interaction prediction with graph neu-ral networks and svm," in 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS), vol. 1. IEEE, 2024, pp. 1–6.
11. M. Azzeh, Y. Elsheikh, A. B. Nassif, and L. Angelis, "Examining the performance of kernel methods for software defect prediction based on support vector machine," Science of Computer Programming, vol. 226, p. 102916, 2023.
12. R. Praveen, P. Pabitha, V. Sakthi, S. Madhavi, and R. V. Sai, "Convolutional neural network and capsule network fusion for effective attrition classification," in 2023 12th International Conference on Advanced Computing (ICoAC). IEEE, 2023, pp. 1–6.