Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

An Efficient Dual Attention Graph-based Recurrent Optimization Model for Software Defect Prediction

Mr. P. Ramesh[1], Dr. Prasath S[2]

[1]Ph.D Research Scholar in Department of Computer Science, VET Institute of Arts and Science (Co-education) College, Thindal, Erode, Tamil Nadu, India

[2]Assistant Professor and Head, Department of Computer Science, VET Institute of Arts and Science (Co-education) College, Thindal, Erode, Tamil Nadu, India

Abstract: Software defect prediction plays a critical role in improving software quality by identifying faulty modules early in the development process. This study aims to address the limitations of existing machine learning and deep learning models, such as their inability to effectively capture structural dependencies, temporal evolution, and high-dimensional feature interactions. To achieve this, a novel model named Dual Attention Graph-based Recurrent Optimization (DAGr-RO) is proposed. The model integrates graph-based learning to represent inter-module relationships, dual attention mechanisms to emphasize important features and structural connections, and recurrent neural networks (LSTM/GRU) to capture temporal patterns across software versions. Additionally, an optimization strategy is employed to dynamically update model parameters and enhance prediction performance. The proposed approach is evaluated using a Kaggle software defect dataset and compared with baseline models including RNN, CNN, Random Forest, and XGBoost. Experimental results demonstrate that DAGr-RO achieves superior performance with an accuracy of 94.5%, precision of 93.2%, recall of 92.6%, and F1-score of 92.9%, outperforming all comparative models. These findings confirm that the integration of graph learning, attention mechanisms, and temporal modeling significantly improves defect prediction. The study concludes that DAGr-RO provides an effective, scalable, and reliable solution for real-world software defect prediction tasks.

Keywords: Attention Mechanism, Deep Learning, Graph Neural Network, Recurrent Neural Network, Software Defect Prediction, Temporal Analysis**.**

1. INTRODUCTION

Software defect prediction has become an important area in software engineering as modern applications grow larger and more complex. Developers and organizations constantly face challenges in identifying faulty modules early, which can otherwise lead to increased development cost and reduced software quality. Traditional testing methods alone are often not sufficient to detect defects efficiently, especially in large-scale systems [1]. Prior studies also highlight the limitations of conventional approaches in handling evolving and domain-specific defect patterns [2]. Furthermore, recent research emphasizes the importance of incorporating advanced metrics and learning techniques to improve prediction reliability [3].

With the advancement of machine learning and deep learning, automated defect prediction techniques have gained significant attention. Models such as RNN have demonstrated the ability to capture temporal dependencies in defect data [4]. Comprehensive surveys further confirm the growing role of deep learning in improving prediction performance [5]. Optimization-based and hybrid learning techniques have also been introduced to enhance feature selection and model accuracy [6]. In addition, classical hybrid approaches like SVM combined with dimensionality reduction remain effective in certain scenarios [7]. Feature engineering and resampling techniques such as SMOTE and RFE have also been widely explored to improve model robustness [8].

However, many of these methods struggle to effectively capture relationships between software components, handle high-dimensional data, and model changes across multiple software versions. Advanced explainability-

based feature selection methods attempt to address these issues but still face scalability concerns [9]. Cloud-based and fusion-based systems improve prediction but introduce complexity in integration and deployment [10]. Comparative studies of machine learning models reveal limitations in generalization across datasets [11].

To address these limitations, this work proposes a new approach called Dual Attention Graph-based Recurrent Optimization (DAGr-RO). The idea is to combine graph learning, attention mechanisms, and temporal modeling to better understand how defects occur and evolve. Ensemble learning techniques with adaptive optimization strategies have shown promise in improving prediction performance [12]. Similarly, stacking-based and tree-based ensemble methods contribute to enhanced classification accuracy [13]. Gradient boosting approaches with optimized hyperparameters further strengthen prediction capabilities [14]. Additionally, cross-project prediction techniques aim to improve generalization across different software datasets [15]. In terms of organization, Section 1 introduces the problem, Section 2 reviews related work, and Section 3 explains the proposed methodology. Section 4 presents results and discussion, and finally, Section 5 concludes the paper with future directions.

2. BACKGROUND STUDY

Salboukh et al. (2026) [1] proposed an RNN-based model incorporating covariates to improve software defect discovery prediction by capturing temporal dependencies. The study highlights the role of external factors in enhancing prediction accuracy. However, it lacks scalability analysis for large datasets and heterogeneous environments. The results show improved performance compared to traditional statistical approaches.

Phung et al. (2025) [2] examined domain-specific error-type metrics in risk-based software fault prediction, emphasizing their impact on prediction accuracy. The study identifies that different domains require tailored evaluation metrics. However, it lacks a generalized framework applicable across domains. Results indicate domain-aware metrics significantly enhance prediction effectiveness.

Mishra and Misra (2026) [3] introduced an adaptive 1-D CNN model with LSelect feature selection to improve software fault prediction accuracy. The approach focuses on reducing feature dimensionality while maintaining performance. However, it shows limitations in handling real-time and diverse datasets. The results demonstrate improved accuracy and reduced computational complexity.

Uddin et al. (2022) [4] proposed a hybrid model combining BiLSTM and BERT for semantic feature extraction in software defect prediction. The approach captures both contextual and sequential information from code. However, it does not integrate structured software metrics effectively. Results show superior performance compared to traditional machine learning methods.

Giray et al. (2023) [5] analyzed the application of deep learning techniques in software defect prediction through a comprehensive study. The work highlights the advantages of deep models over traditional methods. However, it lacks standardized benchmarks for comparison. The study concludes that deep learning improves prediction accuracy with proper tuning.

Anand et al. (2025) [6] developed a defect prediction model using wrapper-based dynamic arithmetic optimization for feature selection. The approach enhances feature relevance and model performance. However, it introduces higher computational complexity. Results indicate improved accuracy and efficient feature reduction.

Mustaqeem and Saqib (2021) [7] proposed a hybrid PCA-SVM model for software defect detection to reduce dimensionality and improve classification. The approach combines statistical and machine learning techniques. However, it struggles with imbalanced datasets and parameter sensitivity. Results show better performance than standalone models.

Ghinaya et al. (2024) [8] utilized SMOTE, RFE, and Random Forest to identify important features in defect prediction and handle class imbalance. The study emphasizes feature selection and data balancing techniques.

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

However, synthetic data generation may lead to overfitting. Results demonstrate improved balanced classification performance.

Hartati et al. (2025) [9] proposed an optimized RFE approach using SHAP with LightGBM for explainable software defect prediction. The model focuses on feature interpretability and accuracy. However, it has high computational requirements and limited real-world validation. Results show improved prediction performance and explainability.

Aftab et al. (2023) [10] introduced a cloud-based defect prediction system using data-level and decision-level machine learning fusion. The approach improves scalability and prediction efficiency. However, it faces challenges related to latency and security in cloud environments. Results indicate enhanced accuracy and distributed system performance.

Table 1: Comparative Analysis of Existing Software Defect Prediction Methods

Author & Year / Ref No	**Concept**	**Research Gap**	**Methods**	**Limitations**	**Result**
Khalid et al. (2023) [11]	Analyzed software defect prediction using machine learning techniques to improve quality.	Lacks deep learning integration and cross-project validation.	Applied multiple ML algorithms for comparative analysis.	Limited scalability and dataset dependency.	ML models improved prediction accuracy.
Tang et al. (2023) [12]	Proposed ensemble learning with Adaptive Variable Sparrow Search Algorithm for prediction.	Limited real-world validation and efficiency analysis.	Combined ensemble models with optimization algorithm.	High computational complexity.	Improved prediction accuracy and optimization.
Alazba & Aljamaan (2022) [13]	Used stacking generalization with optimized tree-based ensembles.	Lack of interpretability and generalization.	Ensemble stacking of decision tree-based models.	Increased complexity and training time.	Achieved better classification accuracy.
AL-Hadidi & Hasoon (2024) [14]	Applied XGBoost with hyperparameter optimization.	Limited hybrid/deep learning exploration.	Extreme Gradient Boosting with tuning.	Overfitting risk and parameter sensitivity.	High accuracy in defect prediction.
Abdu et al. (2025) [15]	Proposed cross-project defect prediction using hybrid software metrics.	Limited scalability and real-time application.	Feature reduction and hybrid metric techniques.	Dataset compatibility issues.	Improved cross-project prediction performance.

This table 1 presents a comparative review of recent software defect prediction approaches based on their concepts, methodologies, research gaps, and limitations. It highlights how different machine learning and optimization techniques improve prediction accuracy while still facing challenges such as scalability, interpretability, and computational complexity. The analysis emphasizes the need for advanced hybrid and deep learning-based models to address these gaps and enhance overall prediction performance.

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

3. PROPOSED METHODOLOGY

The proposed methodology utilizes a Kaggle-based software defect dataset and introduces the DAGr-RO model, which combines graph-based learning, dual attention mechanisms, and recurrent neural networks to effectively capture structural, feature-level, and temporal dependencies. It transforms software metrics into graph representations, applies feature and structural attention to highlight critical information, and leverages RNN (LSTM/GRU) to model defect evolution across versions. The approach is further enhanced through optimization techniques and systematic evaluation, resulting in an accurate, scalable, and efficient defect prediction framework.

3.1 Kaggle Dataset

Dataset: https://www.kaggle.com/datasets/mirzayasirabdullah07/software-defect-prediction-dataset

The dataset used in this study is sourced from Kaggle and contains approximately 60,000 software modules with more than 22 features relevant to defect prediction. It includes various software metrics such as code complexity, lines of code, testing coverage, and development-related attributes, along with a binary target indicating the presence of defects. This dataset is well-suited for training and evaluating machine learning models, as it provides a realistic and comprehensive representation of software quality factors.

3.2 Dual Attention Graph-based Recurrent Optimization (DAGr-RO)

The proposed DAGr-RO algorithm is designed to improve software defect prediction by integrating graph-based learning, dual attention mechanisms, recurrent neural networks and optimization techniques, where graph representation models inter-module dependencies, dual attention (feature and structural) identifies the most relevant metrics and relationships, and recurrent layers (LSTM/GRU) capture temporal evolution of defects across software versions, while an optimization strategy dynamically refines feature weights and model parameters to enhance prediction accuracy and efficiency; the purpose of this algorithm is to overcome limitations of traditional ML/DL models such as poor handling of structured dependencies, temporal dynamics and high-dimensional data and it works by transforming software metrics into graph form, applying attention to highlight critical nodes and features, learning sequential patterns through recurrence and optimizing outputs for classification, enabling near real-time performance through parallel graph computation and efficient attention scoring, while reproducibility is ensured through fixed random seeds, standardized preprocessing, publicly available datasets (e.g., Kaggle) and clearly defined hyperparameters, making the model scalable, consistent and adaptable for real-world defect prediction systems.

$$h_i^{(l+1)} = \sigma(\sum_{j \in N(i)} W^{(l)} h_j^{(l)} + b^{(l)}) \quad (1)$$

In this equation (1), $h_i^{(l)}$ represents the feature vector of node i at layer l, while $N(i)$ denotes its neighboring nodes in the graph. The weight matrix $W^{(l)}$ and bias $b^{(l)}$ are learnable parameters, and σ is a nonlinear activation function. This operation aggregates information from connected modules, allowing the model to learn structural dependencies among software components.

$$\alpha_i = \frac{\exp(\epsilon_i)}{\sum_j \exp(\epsilon_j)} \quad (2)$$

In this equation (2), e_i represents the importance score of the ith feature, and α_i is the normalized attention weight computed using the softmax function. The denominator ensures that all attention values sum to one, creating a probability distribution. This mechanism highlights the most influential software metrics, improving feature selection and reducing noise.

$$e_{ij} = a^T [Wh_i || Wh_j] \quad (3)$$

In this equation (3), e_{ij} defines the attention score between nodes i and j, where W is a transformation matrix and a is a learnable attention vector. The symbol $||$ denotes concatenation of node features after linear

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

transformation. This mechanism evaluates the importance of relationships between software modules, enabling the model to focus on critical interactions.

$$h_t = f(W_h h_{t-1} + W_x x_t + b) \quad (4)$$

In this equation (4), h_t represents the hidden state at time step t, while h_{t-1} is the previous state and x_t is the current input feature vector. The matrices W_h and W_x are weights, and b is the bias term. This formulation captures temporal dependencies in software evolution, allowing the model to learn how defects develop over time.

$$\hat{y} = \sigma(W_o h + b_o) \quad (5)$$

In this equation (5), h is the final learned feature representation from previous layers, while W_o and b_o are output layer parameters. The sigmoid function σ maps the output to a probability between 0 and 1. This equation produces the final prediction indicating whether a software module is defective or not.

Algorithm1: DAGr-RO_Defect_Prediction

Input:

D ← Software metrics dataset (multi-version)

G ← Graph constructed from modules (nodes, edges)

H ← Hyperparameters (learning rate, epochs, hidden units)

seed ← Fixed random seed

Output:

Y_pred ← Defect prediction labels

Begin

Initialize:

Set random seed = seed

Preprocess dataset D (normalization, missing value handling)

Split D into training and testing sets

Graph Construction:

For each software version v in D:

Create node N for each module

Define edges E based on dependencies between modules

Construct Graph G = (N, E)

Feature Embedding:

Convert software metrics into feature vectors F

Map F to graph nodes

Dual Attention Mechanism:

Feature Attention:

For each node i in G:

Compute attention weight α_f(i) over feature set

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

F' = Weighted feature representation

Structural Attention:

For each node i and its neighbors j:

Compute attention weight α_s(i,j)

Aggregate neighbor information:

H_i = Σ α_s(i,j) * F'(j)

Temporal Learning using RNN:

Initialize hidden state h0

For each time step t (software version):

Input H_t into RNN (LSTM/GRU)

Update hidden state:

h_t = RNN(H_t, h_(t-1))

Optimization Strategy:

Initialize model parameters θ

While not converged:

Compute predictions Y_pred

Calculate loss L (e.g., cross-entropy)

Update parameters:

θ = θ - learning_rate * ∇L

Adjust attention weights dynamically

Classification:

Apply sigmoid/softmax on final output

Generate defect labels:

Y_pred = Classify(h_t)

Evaluation:

Compute performance metrics (Accuracy, Precision, Recall, F1-score)

Output results:

Return Y_pred

End

The pseudocode describes the proposed DAGr-RO defect prediction model, which integrates graph-based representation with dual attention mechanisms to capture both feature importance and structural dependencies among software modules. It further incorporates temporal learning using RNN (LSTM/GRU) to model changes across multiple software versions and improve prediction accuracy over time. The model is optimized through iterative training with loss minimization, ultimately producing defect labels and evaluating performance using standard metrics such as Accuracy, Precision, Recall, and F1-score.

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

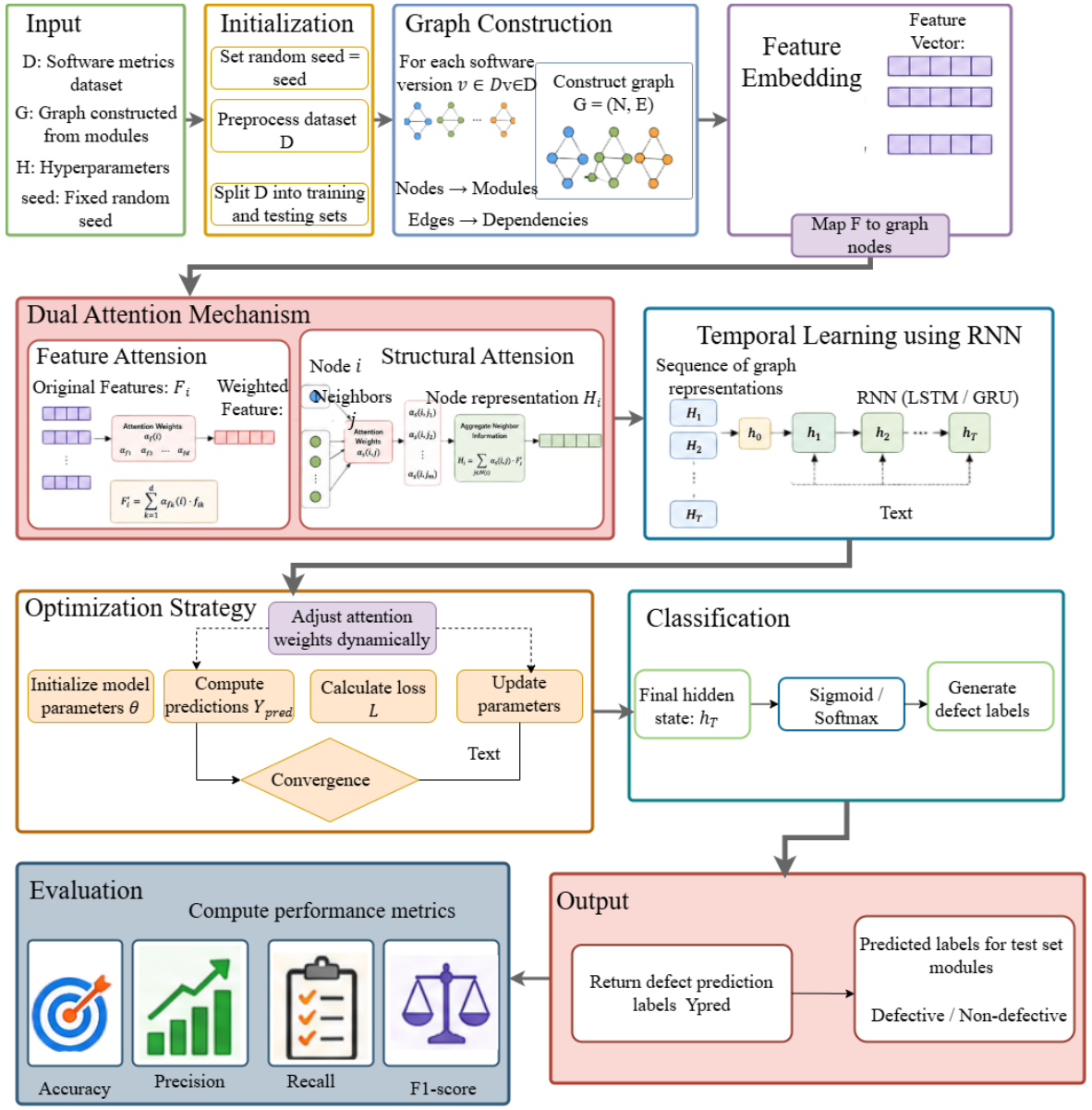

Figure 1: Architecture of Dual Attention Graph-based Recurrent Optimization (DAGr-RO) for Software Defect Prediction

The figure 1 illustrates the complete workflow of the proposed DAGr-RO model, starting from dataset input, preprocessing, and graph construction to represent module dependencies. It integrates dual attention mechanisms (feature and structural) with recurrent neural networks (LSTM/GRU) to capture both important features and temporal evolution of defects. Finally, the model applies optimization and classification to generate defect predictions, evaluated using standard performance metrics such as accuracy, precision, recall, and F1-score.

4. RESULT AND DISCUSSION

The results demonstrate that the proposed DAGr-RO model significantly outperforms existing methods across all evaluation metrics, including Accuracy, Precision, Recall, and F1-Score. Its superior performance is attributed to the integration of graph-based structural learning, dual attention mechanisms, and temporal modeling using RNN. The comparative analysis shows a consistent improvement from traditional and ensemble

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

models to the proposed approach. Overall, the findings confirm that DAGr-RO provides a more accurate and reliable solution for software defect prediction.

4.1 Performance Metrics

4.1.1 Accuracy

To evaluate the effectiveness of the proposed DAGr-RO model, standard classification metrics are used:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy measures the overall correctness of a classification model by calculating the proportion of total predictions that are correct. It considers both positive and negative predictions, making it useful when classes are balanced.

4.1.2 Precision

$$Precision = \frac{TP}{TP + FP}$$

Precision measures how many of the predicted positive instances are actually correct. It is especially important when minimizing false positives is critical.

4.1.3 Recall

$$Recall = \frac{TP}{TP + FN}$$

Recall measures how effectively the model identifies all actual positive instances. It is crucial when minimizing false negatives is important, such as in defect detection.

4.1.4 F1-Score

$$F1 - 2 \cdot \frac{Precision \cdot Recall}{Precision \cdot Recall}$$

The F1-score is the harmonic mean of Precision and Recall, providing a balanced measure of both metrics. It is especially useful when there is an uneven class distribution or when both false positives and false negatives need to be considered.

4.2 Experimental Results

The proposed DAGr-RO model was evaluated using the Kaggle dataset and compared with existing models such as RNN, CNN, Random Forest and XGBoost. The results demonstrate that the proposed model achieves superior performance due to its ability to capture structural, feature-level and temporal dependencies simultaneously.

Table 2: Comparative Performance Analysis of Defect Prediction Models

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
RNN	85.2%	83.1%	82.4%	82.7%
CNN	87.6%	85.9%	84.8%	85.3%
Random Forest	88.9%	87.2%	86.5%	86.8%
XGBoost	90.1%	88.7%	87.9%	88.3%
DAGr-RO (Proposed)	**94.5%**	**93.2%**	**92.6%**	**92.9%**

Journal of Online Engineering Education
ISSN: 2158-9658 Volume: 17 Issue: 2
Article Received: 14 March 2026 **Revised**: 15 May 2026 **Accepted**: 14 June 2026

The table 2 presents a comparative evaluation of defect prediction models using Accuracy, Precision, Recall and F1-Score as performance metrics. It shows a steady improvement from traditional models like RNN and CNN to ensemble methods such as Random Forest and XGBoost. The proposed DAGr-RO model achieves the highest scores across all metrics, demonstrating its superior effectiveness and reliability in software defect prediction.

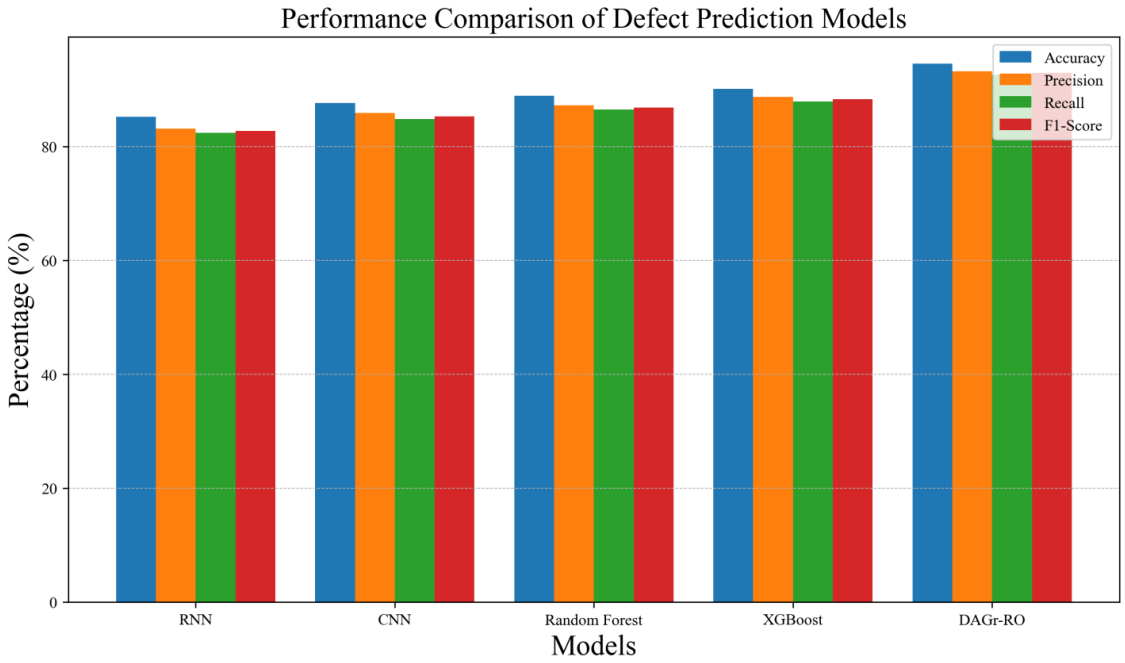

Figure 2: Performance Comparison of Software Defect Prediction Models

The figure 2 compares the performance of different defect prediction models—RNN, CNN, Random Forest, XGBoost and DAGr-RO—across four evaluation metrics: Accuracy, Precision, Recall and F1-Score. It shows a consistent improvement in all metrics from traditional models to advanced approaches, with XGBoost outperforming earlier methods. The proposed DAGr-RO model achieves the highest performance overall, indicating its superior effectiveness in software defect prediction.

5. CONCLUSION

The proposed DAGr-RO model significantly improves software defect prediction by integrating graph learning, dual attention, and temporal modeling. It effectively captures structural dependencies, feature importance, and defect evolution across software versions. The experimental results show superior performance compared to existing machine learning and deep learning models. This demonstrates the model's capability to provide accurate and reliable defect predictions in complex systems. However, the approach involves higher computational cost and requires careful tuning of parameters. In future work, the model can be enhanced with explainable AI, cross-project learning, and real-time deployment for improved scalability and practical applicability.

REFERENCE

[1] Salboukh, F., Silva, P., Nagaraju, V., & Fiondella, L. (2026). Predicting Software Defect Discovery Incorporating Covariates with Recurrent Neural Networks. Quality and Reliability Engineering International, 42(1), 225-236.

[2] Phung, K., Ogunshile, E., & Aydin, M. E. (2025). Domain-specific implications of error-type metrics in risk-based software fault prediction. Software Quality Journal, 33(1), 7. https://doi.org/10.1007/s11219-024-09704-1

[3] Mishra, T., & Misra, S. (2026). Adaptive 1-D CNN Using LSelect Feature Selection for Predicting Software Faults. International Journal of Software Science and Computational Intelligence (IJSSCI), 18(1), 1-23.

[4] Uddin, M. N., Li, B., Ali, Z., Kefalas, P., Khan, I., & Zada, I. (2022). Software defect prediction employing BiLSTM and BERT-based semantic feature. Soft Computing, 26(16), 7877-7891.

[5] Giray, G., Bennin, K. E., Köksal, Ö., Babur, Ö., &Tekinerdogan, B. (2023). On the use of deep learning in software defect prediction. Journal of Systems and Software, 195, 111537.

[6] Anand, K., Jena, A. K., Das, H., Askar, S. S., & Abouhawwash, M. (2025). Software defect prediction using wrapper-based dynamic arithmetic optimization for feature selection. Connection Science, 37(1), 2461080.

[7] Mustaqeem, M., & Saqib, M. (2021). Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection. Cluster Computing, 24(3), 2581-2595.

[8] Ghinaya, H., Herteno, R., Faisal, M. R., Farmadi, A., & Indriani, F. (2024). Analysis of Important Features in Software Defect Prediction Using Synthetic Minority Oversampling Techniques (SMOTE), Recursive Feature Elimination (RFE) and Random Forest. Journal of Electronics, Electromedical Engineering, and Medical Informatics, 6(3), 276-288.

[9] Hartati, H., Herteno, R., Faisal, M. R., Indriani, F., & Abadi, F. (2025). Recursive Feature Elimination Optimization Using Shapley Additive Explanations in Software Defect Prediction with LightGBM Classification. JURNAL INFOTEL, 17(1), 1-16.

[10] Aftab, S., Abbas, S., Ghazal, T. M., Ahmad, M., Hamadi, H. A., Yeun, C. Y., & Khan, M. A. (2023). A cloud-based software defect prediction system using data and decision-level machine learning fusion. Mathematics, 11(3), 632.

[11] Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software defect prediction analysis using machine learning techniques. Sustainability, 15(6), 5517.

[12] Tang, Y., Dai, Q., Yang, M., Du, T., & Chen, L. (2023). Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. International Journal of Machine Learning and Cybernetics, 14(6), 1967-1987.

[13] Alazba, A., &Aljamaan, H. (2022). Software defect prediction using stacking generalization of optimized tree-based ensembles. Applied Sciences, 12(9), 4577.

[14] AL-Hadidi, T. N., &Hasoon, S. O. (2024). Software defect prediction using extreme gradient boosting (XGBoost) with optimization hyperparameter. Al-Rafidain Journal of Computer Sciences and Mathematics, 18(1), 22-29.

[15] Abdu, A., Zhai, Z., Abdo, H. A., Lee, S., Al-masni, M. A., Gu, Y. H., & Algabri, R. (2025). Cross-project software defect prediction based on the reduction and hybridization of software metrics. Alexandria Engineering Journal, 112, 161-176.