Journal of Technological Innovations

Est. 2020

Predictive Software Defect Identification with Adaptive Moment Estimation based Multilayer Convolutional Network Model

Mohnish Neelapu
Email id: neelapu1001@gmail.com

Abstract

The practice of predicting software errors in quality assurance now experiences a significant advancement through AI automation. The system uses natural language processing together with data analytics and machine learning techniques to examine historical records in order to make defect identification. The existing defectidentification models struggle with various challenges that stem from noisy data and class imbalanced datasets and complex pattern recognition tasks because their performance deteriorates. In this research develop a new model Adaptive Moment Estimation based Convolutional Neural network –Multi Layer Perception model (Adam based CNN-MLP), for defect identification as it combines CNN features extraction power with adaptive MLP identification capabilities. The system extracts essential data points from unprocessed information while developing general skills through its ability to detect intricate patterns between software faults. The CNN segment first extracts spatial patterns from the data before the MLP component uses identification abilities to analyze high-level dependencies. The combination of AI advanced features creates an optimal solution which enables efficient and accurate scaling of software defect identification while expanding AI quality management capabilities in software engineering

Keywords: Predictive software Defect Identification, Convolutional Neural Network, Multi-Layer Perceptron, hybrid Convolutional Neural Network and Multi-Layer Perceptron.

Introduction

Predictive Software Defect Identification system uses AI-based Quality Engineering to resolve the testing constraints that exist in contemporary software development practices [1]. Standard quality assurance faces difficulties in meeting application complexity requirements and security standards because organizations now widely use cloud computing together with micro services and IoT technology along with Artificial Intelligence capabilities [2]. Current business needs more rapid product releases as well as real-time application monitoring to maintain competitiveness yet standard defect detection methods cannot meet these requirements [3]. AI-based identification models which use machine learning and deep learning technology help identify defects in advance through the analysis of historical information and code structures with system activities [4]. The intelligent automation boosts product reliability through faster development times while ensuring top-quality software outcomes which makes predictive defect identification vital for software engineering [5].

The testing processes that heavily depend on manual testing combined with scripted test cases commonly fail to detect advanced bugs before the development lifecycle reaches its latter phases [6-7]. The manual testing methods used by these systems become ineffective for processing extensive applications because they require excessive time and resources. Inaccurate identification emerge from models designed to detect defects because faulty historical data that lacks consistency or completeness becomes a problem for identification accuracy [8]. When AI models operate as black box systems it creates difficulties for developers to understand how identification were made so they become doubtful about system reliability [9]

A more integrated method should be used to eliminate these system limitations. The accuracy of AI identificationdepends on enhancing the data quality utilized for training AI models thus requiring complete accurate and consistent defect data labels for better outcomes [10-12]. Hybrid automation solutions which unite AI-based systems and human testers solve interpretability problems because they enable testers to see how AI makes its identification. AI model efficiency requires continuous observation and repeated training to allow them to recognize emerging patterns and trends within the software development process [13-14]. AI-powered defect identification systems gain robustness together with reliability through the implementation of these strategies which leads to

transparent improvements in software quality management [15].

- **Adam based CNN-MLP:**The proposed Adam-based CNN-MLP model enhances software defect identification by integrating CNN for effective feature extraction and MLP for robust classification. Unlike existing models, it addresses challenges such as high computational complexity, overfitting, and slow convergence. LDA preprocessing improves class separability and reduces redundant features, optimizing computational efficiency. The Adam optimizer dynamically adjusts learning rates, ensuring stable convergence and reducing overfitting. This approach enhances accuracy, robustness, and efficiency, making it a more effective solution for software defect identification.

Literature Review

Ahmed Abdu et al. [1] created Ensemble Convolutional Neural Network deep learning which serves as an approach for software defect identification. This research strives to develop a dependable approach for software defect identification within specific projects to aid engineers during resource management for high quality software delivery. Predictive results improve due to this method because it detects both numerical data patterns and semantic comparisons. The approach has limited effectiveness because semantic feature extractions using Word2Vec might not work reliably or accessibly in particular situations. The study by Mohd Mustaqueem et al. [2] developed the sampling, identification, and Analysis Model (SPAM) with Explainable Artificial Intelligence system as a hybrid deep learning framework to predict software defects with XAI approaches to improve performance visibility. The SPAM-XAI model provides transparent insights into how features relate to error status which leads to better understandable identification. The method enhances transparency together with raising identification accuracy. The non-linear models lead to harder interpretation and higher computational complexity serves as the main disadvantages. Andressa Borre et al. [3] developed hybrid Convolutional Neural Network (CNN)-Long Short-Term Memory (LSTM). This method merges CNN with LSTM network for its operation. The method functions to anticipate electrical machine breakdowns and handles data-related uncertainties effectively. High accuracy and Time-Series data suitability stand out as key advantages of this method while its main drawback includes high computational complexity, Over fitting and needs extensive data quantities. The research by Sajid Mehmood et al. [4] presents a detection system for distributed denial of service attacks in Software-Defined Networks (SDN) through optimizing a combination of Convolutional Neural Networks (CNN) and Multi-Layer Perceptron (MLP) with an optimizer-equipped approach. The research presents possible approaches for future SDN network security development. The usage advantages of this method include increased scalability as well as better detection precision. The main drawbacks of this approach involve complex computations and unclear model explanations.

A.Challenges

- Real-time deployment becomes complicated because the unified structure of CNN, LSTM, and attention mechanisms creates high computational complexity [1].
- The model needs extensive labeled data for effective training but such resources might be scarce particularly when detecting faults [2].
- Overfitting due to the large number of parameters in CNN and LSTM models, particularly with small datasets. [2].
- Real-time detection could be affected by the Adam based CNN-MLPmodel and optimizer because they increase computational expenses. [3].
- The detection system requires extensive labeled training data to function effectively yet such data might prove difficult to acquire for DDoS detection purposes [4].
- The combination of software metrics complexity with the optimizer creates computational challenges that make model convergence difficult to handle. [5].
- The complex CNN-MLP hybrid approach makes it challenging to determine precise defect identification reasons because of its limited interpretability capabilities [6].
- The uneven distribution of defective and non-defective instances within the data causes identification bias because of data imbalance. [6].
- The lack of interpretability in CNN and MLP models creates difficulties for software engineers to trust their identification because the models operate as black boxes [6].

PROPOSED METHODOLOGY FOR PREDICTIVE SOFTWARE DEFECT IDENTIFICATION WITH ADAPTIVE MOMENT ESTIMATION BASED CNN-MLP MODEL

This research proposes a predictive software defect identification model by utilizing anAdam based CNN-MLP approach, where the input is collected from a software defect identification dataset(https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction) that includes software metrics such as lines of code, complexity measures, and other relevant features. The dataset requires Linear Discriminant Analysis (LDA) preprocessing before it can proceed. By employing LDA the model achieves better identification results because this method preserves class separability in reduced feature spaces. The Adam based CNN-MLPmodel receives preprocessed data for its software Defect Identification. The Adam based CNN-MLPmodel uses the best qualities of both its components to enhance software defect identification accuracy along with robustness performance. The model

optimization process uses Adam as an optimization algorithm which adjusts individual parameter learning rates during training. Adam employs first and second gradient moments during training for stable convergence while boosting the Fig. 1. Input from software defect identificationdataset. Preprocessing using linear discriminant analysis.

efficiency of the overall process. The predictive software Defect Identification system utilizes Adaptive Moment Estimation based Adam based CNN-MLPmodel as shown in

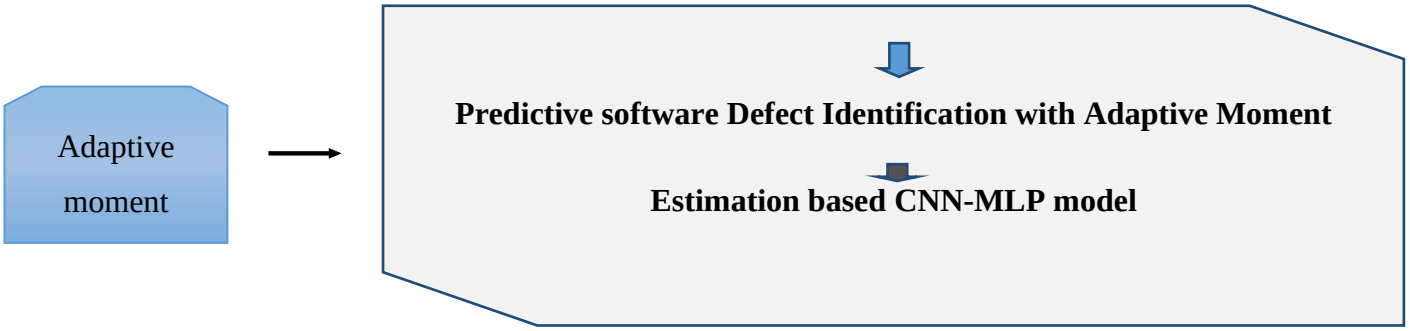


Fig. 1. Architecture for Predictive software Defect Identification with Adam based CNN-MLPmodel

A. Input Collection from Software Defect identification Dataset:

The main dataset derives from Software Defect identification Dataset https://www.kaggle.com/datasets/semustafacevik/software-defect-identification and contains static code attributes alongside maintainability index, cyclomatic complexity, lines of code and code chum features. These performance indicators present information about defect-prone modules in the gathered software project data. Working and faulty outcomes in the data help the model identify patterns that relate to software quality levels.

$$A = \sum_{a=1}^{n} S_a, \quad (1)$$

Where, A denotes the dataset, S_a denotes the number of metrics present in the dataset with values ranges from 1 to n.

B. Preprocessing Using linear discriminant analysis:

LDA is used as a preprocessing step in software defect identification to reduce dimensionality while preserving class separability. The dataset includes features like lines of code, complexity measures, and coupling metrics, which may introduce redundancy. LDA projects data onto a lower-dimensional space to maximize the distinction between defective and non-defective components, unlike PCA, which focuses on variance. This enhances feature representation, improves generalization, and accelerates convergence when applied before training the CNN-MLP model, ultimately boosting defect identification accuracy.

$$X_{LDA} = W^T S_a^* \quad (2)$$

C. Hybrid Convolutional -Multi layer model for software defect identification:

The preprocessed output forms the input for the Adam based CNN-MLPmodel for software defect identification.The Adam based CNN-MLPmodel combines deep learning features of CNN with MLP's classification through a unified framework to achieve higher predictive accuracies. Data patterns become detectable through CNN because the framework uses convolutional filters together with ReLU activations and pooling layers to purify critical information within the data. The processed features move through MLP which enables effective defect classification to reach better predictive modeling performance. The main function of CNNs involves spatial data processing yet they need a classifier to generate final identification. MLP serves as the classification decision engine because it utilizes fully connected layers to process extracted features. A one-dimensional vector derived from the CNN output moves to the MLP before it utilizes Sigmoid or SoftMax activation functions to calculate defect probabilities. The combination of CNN and MLP within this hybrid model strengthens defect identification software by extracting effective features before performing exact classification operations. Adam based CNN-MLPmodel receives its input through combined output data from both CNN and MLP. The equation for Adam based CNN-MLPmodel is calculated by using the inputs are,

$$X_{CNN} = \mathrm{ReLU}(\mathrm{Conv\ 2D}(X_{LDA}) + B_{CNN}) \quad (3)$$

$$X_{flattened} = \mathrm{Flatten}(X_{CNN}), \quad (4)$$

$$X_{MLP} = \sigma(W_2 . \mathrm{MLP}(X_{flattened}) + b_2), \quad (5)$$

The output equation be,

$$H_{CNN-MLP} = \mathrm{concat}(X_{CNN}, X_{MLP}) \quad (6)$$

The final output $H_{CNN-MLP}$ represents the probability of a software component being defective or not. Fig. 2 illustrates the Architecture for the proposedAdam based CNN-MLP model.

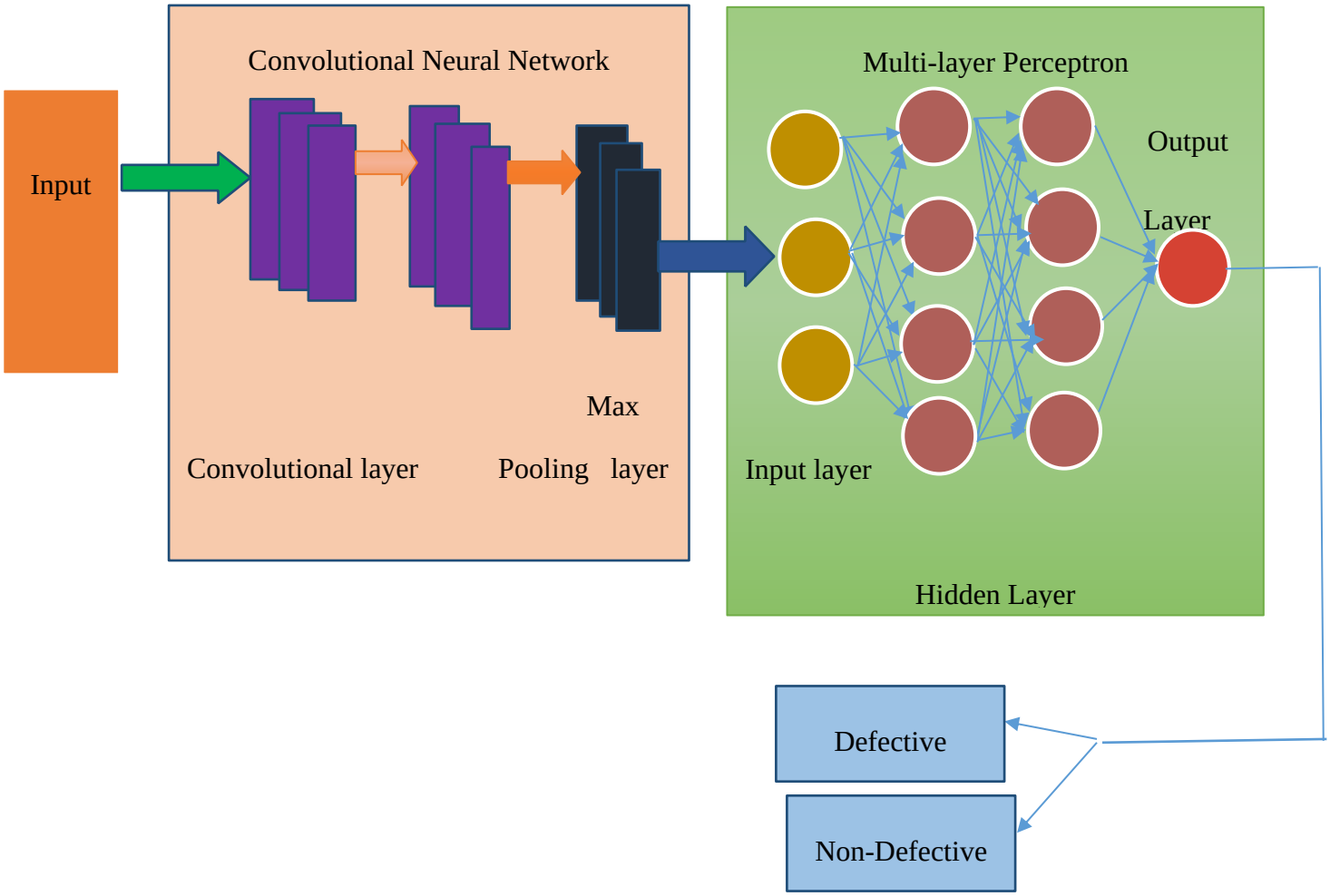


Fig. 2. Architecture for the proposed Adam based CNN-MLP model.

IV. Proposed adaptive moment Estimation Algorithm:

The Adam algorithm optimizes the CNN-MLP model by dynamically tuning the weights and biases during training. These functions allow the model to perform updates that merge Momentum-based Gradient Descent with RMSprop advantages. The Adam optimizer serves as an effective tool for training deep networks especially when used with the Adam based CNN-MLPmodel for software defect identification**.**

A. parameters initialization

The model parameters consisting of weights (W) and biases (b) receive initial values to extract meaningful patterns from the dataset for better predictive accuracy. Adam optimizer manages two fundamental moment estimates through m_t that follows an exponential weighted average of past gradients and v_t that tracks an exponential weighted average of past squared gradients to control update scaling. Adam needs a learning rate (η) to control step size together with decay rates (β1 and β2) to balance past contributions and a small constant (ϵ) to prevent division by zero for achieving stable and efficient parameter updates.

B. Gradient of loss function

The calculation of gradient for the loss function occurs with respect to model parameters during every iteration. The loss function determines model defect identification accuracy while its gradient indicates the parameter adjustment direction.

$$g_t = \nabla \text{Loss}(W, b) \quad (7)$$

Where $g_t$represents the gradient at iteration t, this gradient indicates whether parameters should be increased or decreased to minimize the loss.

C. First Moment Estimate

Adam implements the first moment estimate to build a history of gradient accumulation which stabilizes its update functions. Adam performs calculations using smoothed gradient values obtained through an exponential moving average instead of raw gradient data,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad (8)$$

Where,

- The term m_t helps smooth out noisy updates, preventing sudden drastic changes in weights.
- The hyper parameter β_1 determines how much of the past gradients contribute to the update.

D. Second Moment Estimate

Adam updates differ from standard gradient descent because it tracks both gradient magnitude and its time-dependent second moment calculation. The method controls big update variations to keep learning operations stable.

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g^2{}_t, \quad (9)$$

Where,

- v_t be the updated second moment estimation,
- β_2 controls how much past squared gradients influence the current update,
- $g^2{}_t$ represents the square of current gradient.

E. Bias correction for first and second moments

The starting values for first and second moment estimates are zero at training onset. The early updating values exhibit zero bias because the initial estimates start at zero which leads to improper update calculations. The implementation of bias correction produces unbiased estimates that fix the initial issue with the moments.

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad (10)$$

Adam does not rely on overly small or biased moment estimates in the initial stages, leading to more stable training.

F. Model parameters using Adam update rule

The model receives its updates based on the unbiased moment estimates we have obtained. The learning rate falls under dynamic adjustment per parameter during this step which enhances the training process by avoiding overcorrection.

$$W = W - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t}+\varepsilon}, \; b = b - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t}+\varepsilon} \quad (11)$$

Where,

- W and b are the weights and biases being optimized,
- H be the learning rate,
- $\widehat{m}_t$ and $\widehat{v}_t$ are the corrected moment estimates obtained in the previous step.

G. Check Stopping Criteria

Adam executes multiple optimization steps until it fulfils one of its predefined stopping criteria. The algorithm terminates after completing the predefined training epoch count which represents the first termination condition. The model stops training when loss convergence occurs because the changes in the loss function become minimal which indicates the model has achieved its optimal solution. The training process stops when validation accuracy remains unchanged across multiple iterations in order to avoid Overfitting.

H. Output the Optimized model

After completing all optimization steps, the best set of parameters (weights and biases) are obtained as,

$$W^*, b^*, \quad (12)$$

Where,

✓ W^*, b^* are the final optimized weight and bias value.

These optimized parameters are now used in Adam based CNN-MLPmodel for software defect identification, ensuring that the model generalizes well to new data and achieves high accuracy.

TABLE I

PSEUDOCODE FOR THE ADAM ALGORITHM:

S.No	Pseudocode for the Adam Algorithm
1	**Initialize parameters:** Set weights W, biases b, first moment m_t =0, second moment v_t=0, learning rate η, and hyperparameters $\beta_1, \beta_2, \epsilon$.
2	**Compute gradients:** Calculate the gradient of the loss function, $g_t = \nabla \text{Loss}(W, b)$.
3	**Update first moment estimate:** Compute exponentially weighted moving average of gradients, $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$.
4	**Update second moment estimate:** Compute exponentially weighted moving average of squared gradients, $v_t = \beta_2 v_{t-1} + (1-\beta_2) g^2{}_t$.
5	**Bias correction:** Adjust moment estimates to correct initialization bias, $\widehat{m}_t = \frac{m_t}{1-\beta_1^t}$, $\widehat{v}_t = \frac{v_t}{1-\beta_2^t}$.
6	**Parameter update:** Update weights and biases using the Adam update rule, $W = W - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t}+\varepsilon}$, $b = b - \frac{\eta \widehat{m}_t}{\sqrt{\widehat{v}_t}+\varepsilon}$.
7	**Stopping criteria:** Repeat 4.2 to 4.6 until convergence is achieved.
8	**Output optimized parameters:** Return the final updated weights W and biases b that minimize the loss function.

Result and discussion

AnAdam based CNN-MLPmodel for predicting software defects developed in this study. By comparing its performance to that of other top models, its effectiveness is evaluated.

A. Experimental Setup

The software defect identification Experiment is carried out using a Python script (version 3.7.6) on a Windows 10 OS with 8GB of RAM.

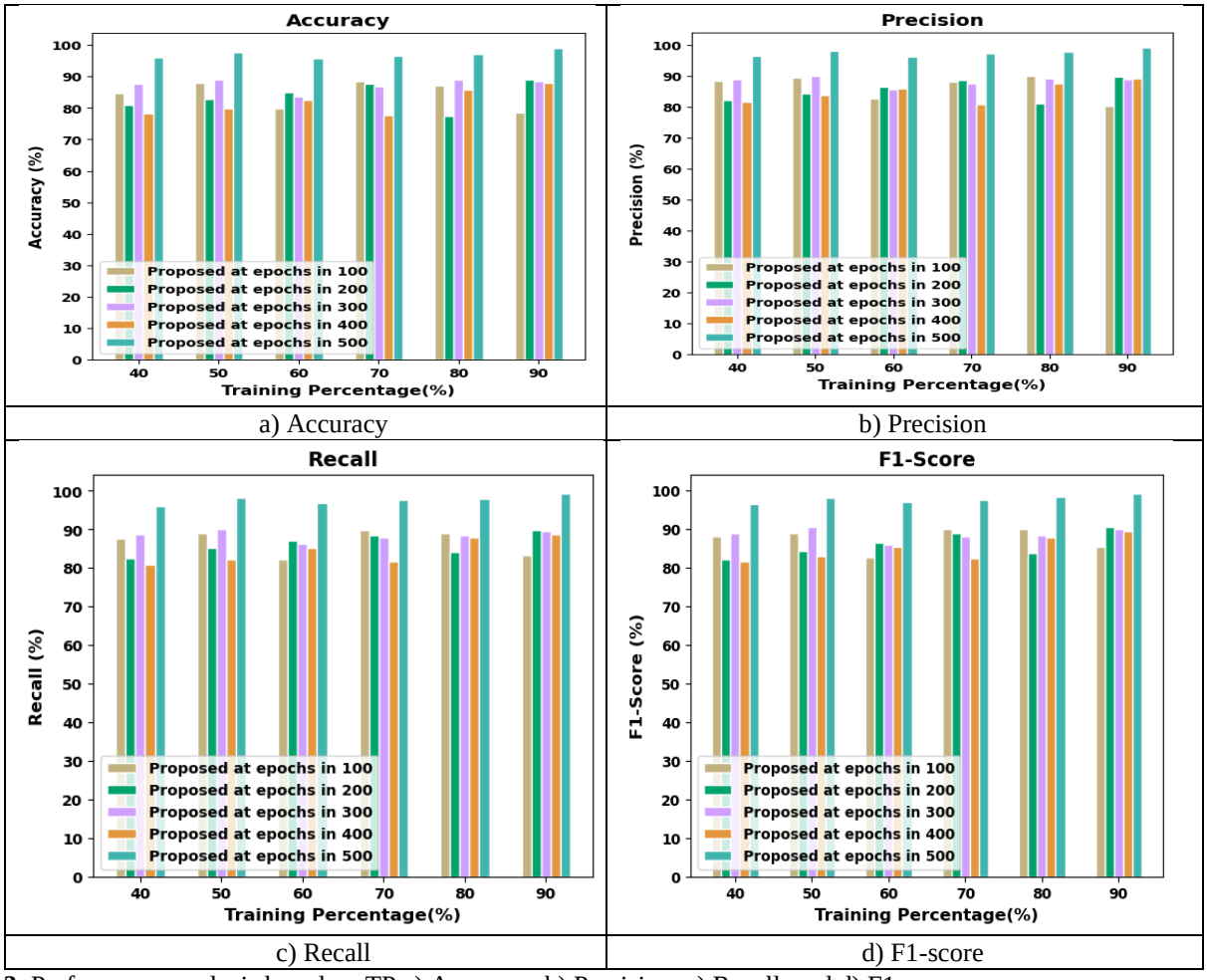
B. Dataset description:

Software Defect identification Dataset description [21]:

The Kaggle Software Defect identification Dataset enables evaluations through software module measurement data composed of lines of code (LOC), cyclomatic complexity, depth of inheritance tree (DIT), coupling between objects (CBO) and other structural elements that determine defect sensitivity. The model contains two possible outcomes which are defective (1) and non-defective (0) thus making it appropriate for binary classification tasks. The dataset covers several software versions which enable researchers to study model generalization capabilities. The identification accuracy suffers due to the class imbalance problem that exists when defective modules occur less frequently than non-defective modules. The data collection serves as a fundamental resource for developing early defect detection systems that improve software quality together with reliability.

C. Performance Analysis based on TP:

Fig. 3 demonstrates the effectiveness of the Adam based CNN-MLPmodel in software defect identification by analyzing its performance across varying epochs (100, 200, 300, 400, and 500) while maintaining a TP of 90. In Fig. 3a, it is evident that the accuracy for these epochs attained remarkable levels: 79.5%, 89.5%, 83.5%, 80%, and 97%. Similarly, Fig. 3b indicates that the Adam based CNN-MLPmodel reached its higher precision scores of 79.5%, 89.5%, 83.5%, 80%, and 97% also corresponding to the same TP of 90. Additionally, Fig. 3c presents findings for the same epochs, showcasing the higher recall rates of 80%, 89%, 85%, 81%, and 97.5% at a TP of 90. Lastly, Fig. 3d reveals the f1-score results for the Adam based CNN-MLP model at a TP of 90, highlighting peak value of 85.3%, 90.6%, 89.9%, 89.5%, and 99.1%.



a) Accuracy b) Precision

c) Recall d) F1-score

Fig. 3. Performance analysis based on TP a) Accuracy, b) Precision, c) Recall, and d) F1-score.

D. Comparative Methods

To emphasize the accomplishments of the Adam based CNN-MLPmodels, a comparison was done. This investigation employed a number of techniques, such as Ensemble CNN [1], SPAM-XAI method [2], hybrid CNN-LSTM model [3], CNN with optimizer –equipped approach [4].

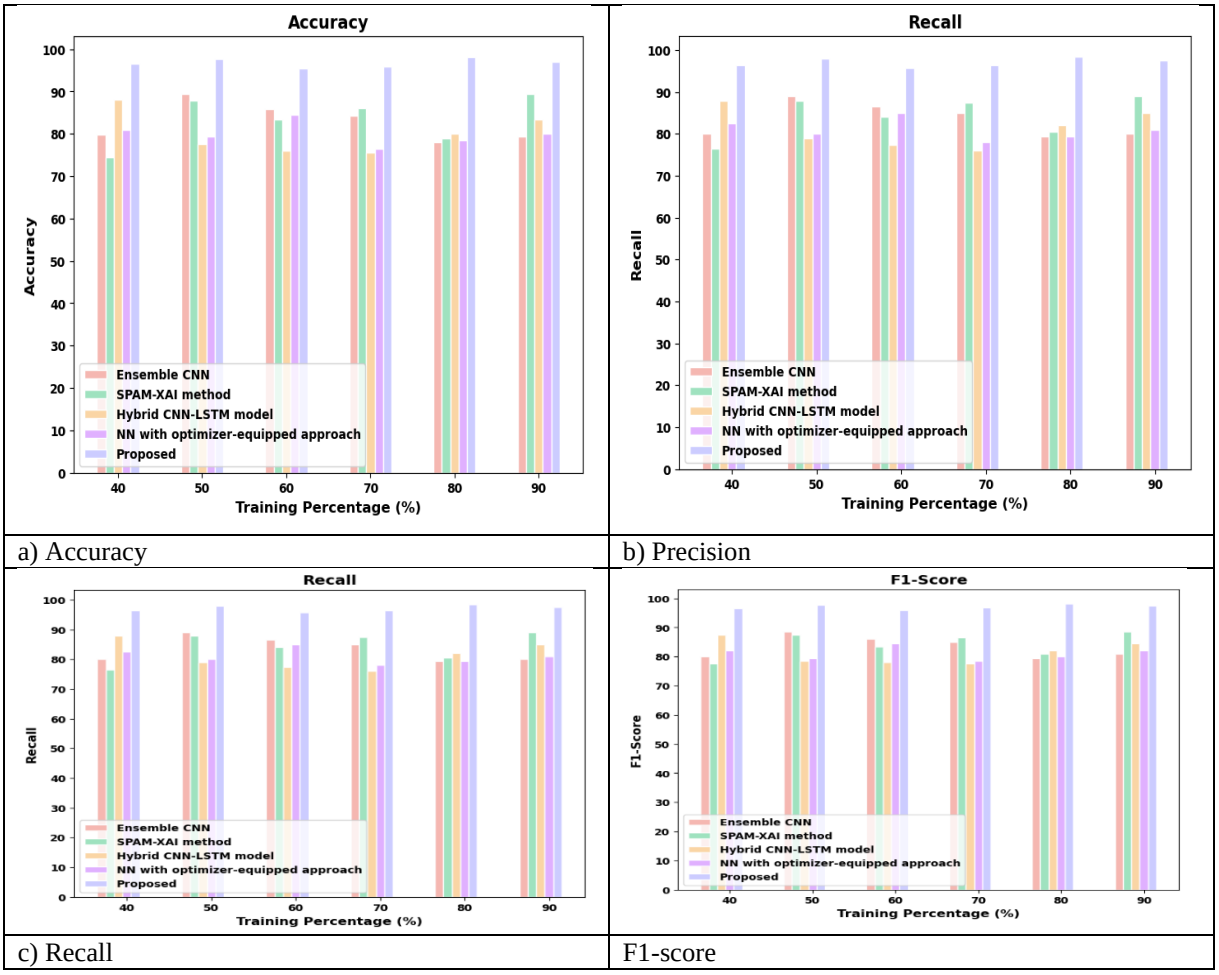
1) Comparative analysis based in TP

The Adam based CNN-MLP model performed better than the CNN with optimizer –equipped model at predicting software defects at a TP of 90 through an improvement of 17.52% which reached a peak accuracy of 97% as illustrated in fig. 4a.

Fig. 4b demonstrates how the Adam based CNN-MLP provides superior forecasting capabilities for software defects than the CNN with optimizer –equipped model while reaching a 97% precision level at a TP of 90. This leads to a 17.52% performance advantage.

The Adam based CNN-MLP model surpassed the CNN with optimizer –equipped model by 16.92% in its software defect identification while attaining a maximum recall of 97.5% at a TP of 90.

Fig. 4d shows that the Adam based CNN-MLP model outperforms the DE model for software defect identification through an f1-score of 97.5% at a TP of 90 while providing 15.89% more performance than the CNN with optimizer –equipped model.



a) Accuracy | b) Precision

c) Recall | F1-score

Fig. 4. Comparative Analysis based on TP a) Accuracy, Precision, c) Recall, and d) F1-score.

E. Comparative discussion table during TP 90

Existing software defect identification models, such as Ensemble CNN, SPAM-XAI, Hybrid CNN-LSTM, and CNN with Optimizers, face challenges like high computational complexity, feature extraction difficulties, overfitting, and slow convergence. Ensemble CNN improves accuracy but increases complexity, SPAM-XAI enhances interpretability at the cost of performance, Hybrid CNN-LSTM captures sequential dependencies but struggles with vanishing gradients, and CNN

with Optimizers lacks adaptability in learning rate adjustments. The proposed Adam-based hybrid CNN-MLP model addresses these issues by combining CNN for feature extraction and MLP for classification, ensuring efficient learning and decision-making. The comparative discussion table based on TP 90 can be found in Table II.

TABLE II

Model	TP 90			
	Accuracy	**Precision**	**Recall**	**F1-score**
Ensemble CNN	79.5	79.5	80	97.5
SPAM-XAI Method	89.5	89.5	89	82
Hybrid CNN-LSTM method	83.5	83.5	85	84.5
CNN with Optimizer-equipped approach.	80	80	81	88.5
Proposed Adam based hybrid CNN-MLP model.	97	97	97.5	97.5

COMPARATIVE DISCUSSION TABLE FOR TP 90.

Conclusion

The proposed Adam based hybrid CNN-MLP model establishes an AI-based Quality engineering system which makes exact decisions about software defect identification. The Adam based hybrid CNN-MLP Framework begins its process by improving data quality through noise reduction. The Hybrid CNN-MLP model receives the preprocessed dataset for processing at the same time it utilizes an adaptive learning rate mechanism which speeds up convergence and boosts model stability during training sessions. The research addresses model limitations by providing exact defect monitoring and better software quality through its solution.

References

[1]A. Abdu, Z. Zhai, H. A. Abdo, R. Algabri, M. A. Al-Masni, M. S. Muhammad, and Y. H. Gu,"Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model," Scientific Reports, vol. 14, no. 1, pp. 14771,2024.

[2]M. Mustaqeem, S. Mustajab, M. Alam, F. Jeribi, S. Alam, and M. Shuaib, "A trustworthy hybrid model for transparent software defect prediction: SPAM-XAI," Plos one, vol. 19, no. 7, pp. e0307112,2024.

[3] A. Borré, L. O. Seman, E. Camponogara, S. F. Stefenon, V. C. Mariani,andL. D. S. Coelho, "Machine fault detection using a hybrid CNN-LSTM attention-based model," Sensors, vol. 23, no. 9, pp. 4512,2023.

[4] S. Mehmood, R. Amin, J. Mustafa, M. Hussain, F. S. Alsubaei, and M. D. Zakaria,"Distributed Denial of Services (DDoS) attack detection in SDN using Optimizer-equipped CNN-MLP," PloS one, vol. 20, no. 1, pp. e0312425,2025.

[5] J. A. Wass, ''WEKA machine learning workbench,'' Sci. Comput., vol. 24, pp. 1–4, Jan. 2007.

[6] A. Ahmad, ''Use of minitab statistical analysis software in engineering technology,'' in Proc. ASEE Annu. Conf. Expo., pp. 1–10,2019.

[7] L. Niu, ''A review of the application of logistic regression in educational research: Common issues, implications, and suggestions,'' Educ. Rev., vol. 72, no. 1, pp. 1–27, 2018.

[8] R. R. Bouckaert, ''Bayesian network classifiers in weka for version 3-5-7,'' Artif. Intell. Tools, vol. 11, no. 3, pp. 369–387, 2008.

[9] G. Kaur, and A. Chhabra, ''Improved J48 classification algorithm for the prediction of diabetes,'' Int. J. Comput. Appl., vol. 98, no. 22, pp. 13–17, Jul. 2014.

[10] J. P. G. Sterbenz, et al., ''Redundancy, diversity, and connectivity to achieve multilevel network resilience, survivability, and disruption tolerance invited paper,'' Telecommun. Syst., vol. 56, pp. 17–31, 2014. doi: 10.1007/s11235-013-9816-9.

[11] Khalid, Aimen, Gran Badshah, Nasir Ayub, Muhammad Shiraz, and Mohamed Ghouse, "Software defect prediction analysis using machine learning techniques," Sustainability,vol. 15, no. 6, pp. 5517,2023.

[12] Singh, Praman Deep, and Anuradha Chug, "Software defect prediction analysis using machine learning algorithms," In 2017 7th international conference on cloud computing, data science & engineering-confluence, IEEE,pp. 775-781, 2017.

[13] Prabha, C. Lakshmi, and N. Shivakumar, "Software defect prediction using machine learning techniques," In 2020 4th International conference on trends in electronics and informatics (ICOEI)(48184), IEEE, 2020, pp. 728-733.

[14] Singh, Praman Deep, and Anuradha Chug, "Software defect prediction analysis using machine learning algorithms." In

2017 7th international conference on cloud computing, data science & engineering-confluence, IEEE, 2017, pp. 775-781.
[15] Rivas, Pablo, Javier Orduz, Tonni Das Jui, Casimer DeCusatis, and Bikram Khanal, "Quantum-enhanced representation learning: A quanvolutional autoencoder approach against ddos threats," Machine Learning and Knowledge Extraction,vol. 6, no. 2, pp. 944-964,2024.
[16] Yang, Xingguang, Huiqun Yu, Guisheng Fan, and Kang Yang, "DEJIT: a differential evolution algorithm for effort-aware just-in-time software defect prediction," International Journal of Software Engineering and Knowledge Engineering,vol. 31, no. 03, pp. 289-310,2021.
[17] Khleel, Nasraldeen Alnor Adam, and Károly Nehéz, "A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method," Journal of Intelligent Information Systems,vol. 60, no. 3, pp. 673-707,2023.
[18] Mehmood, Iqra, Sidra Shahid, Hameed Hussain, Inayat Khan, Shafiq Ahmad, Shahid Rahman, Najeeb Ullah, and Shamsul Huda, "A novel approach to improve software defect prediction accuracy using machine learning," IEEE Access,vol. 11, pp. 63579-63597,2023.
[19] J. Cai, J. Luo, S. Wang, and S. Yang, ‘‘Feature selection in machine learning: A new perspective,’’ Neurocomputing, vol. 300, pp. 70–79, Jul. 2018.
[20] K. P. Singh, N. Basant, and S. Gupta, ‘‘Support vector machines in water quality management,’’ Analytica Chim. Acta, vol. 703, no. 2, pp. 152–162, Oct. 2011.
[21] Software Defect Prediction Dataset link:
https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction