IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 January 2024, accepted 2 April 2024, date of publication 16 April 2024, date of current version 24 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3388489

RESEARCH ARTICLE

LCNN: Lightweight CNN Architecture for Software Defect Feature Identification Using Explainable AI

MOMOTAZ BEGUM[1], MEHEDI HASAN SHUVO[1], (Member, IEEE), MOSTOFA KAMAL NASIR[2], AMRAN HOSSAIN[1], (Member, IEEE), MOHAMMAD JAKIR HOSSAIN[3], (Member, IEEE), IMRAN ASHRAF[4], JIA UDDIN[5], AND MD. ABDUS SAMAD[4], (Member, IEEE)

[1]Department of Computer Science and Engineering, Dhaka University of Engineering & Technology (DUET), Gazipur 1707, Bangladesh
[2]Department of Computer Science and Engineering, Mawlana Bhashani Science & Technology University, Tangail 1902, Bangladesh
[3]Department of Electrical and Electronic Engineering, Dhaka University of Engineering & Technology (DUET), Gazipur 1707, Bangladesh
[4]Department of Information and Communication Engineering, Yeungnam University, Gyeongsan-si 38541, South Korea
[5]AI and Big Data Department, Endicott College, Woosong University, Daejeon 34606, South Korea

Corresponding authors: Md. Abdus Samad (masamad@yu.ac.kr) and Jia Uddin (jia.uddin@wsu.ac.kr)

This research is funded by Woosong University Academic Research 2024.

ABSTRACT Software defect identification (SDI) is a key part of improving the quality of software projects and lowering the risks that along with maintenance. It does identify the software defect causes that have not been reached yet to get sufficient results. On the other hand, many researchers have recently developed several models, including NN, ML, DL, advanced CNN, and LSTM, to enhance the effectiveness of defect prediction. Due to an insufficient dataset size, repeated investigations, and no longer appropriate baseline selection, the research on the CNN model was unable to produce reliable results. In addition, XAI a well-known explainability approach creates deep models in computer vision, as well as successfully handles the software defect prediction that is easy for humans to understand. To address these issues, firstly we have used SMOTE for preprocessing which was collected from the NASA repository; categorical and numerical data. Secondly, we have experimented with software defect prediction using 1D-CNN and 2D-CNN named lightweight CNN (LCNN). Subsequently, evaluation we have employed a 100-repetition holdout validation. For the cross-validation setup, we utilized the 1D-CNN model was 20×1, and for the 2D-CNN model, it was $4 \times 5 \times 1$. After that, the results of the experiment were compared and assessed in terms of accuracy, MSE, and AUC. The result shows that 2D-CNN shows 1.36% better contrast with 1D-CNN. Thirdly, we have conducted research on the identification of software defect features via LIME and SHAP in XAI stand as state-of-the-art techniques. However, we cannot use 2D-CNN because it involves more complex relationships, making it challenging to create transparent explanations. That is why we have realized that 1D-CNN will superior result to explain the root cause of software feature identifications. Finally, LIME provides accurate visualization of software defect features in contrast with SHAP, as well as it helps the stakeholders of the software industry easily find actual root causes of software defect identification.

INDEX TERMS Software defect identification (SDI), explainability, 1D-CNN, 2D-CNN, CNN model, deep learning, SHAP, LIME.

I. INTRODUCTION

In today's world, the use of software is increasing day by day in different domains such as increased efficiency and productivity, enhanced communication and collaboration, entertainment and leisure, innovation and technological advancement, accessibility, and global connectivity. Software defects, ranging from coding errors to design flaws, can have far-reaching consequences, leading to system failures, security breaches, and financial losses. That is why the identification and mitigation of software defects is a pivotal issue for ensuring reliable and safe software. On the other hand, the field of software engineering has long recognized the importance of early fault detection and

The associate editor coordinating the review of this manuscript and approving it for publication was Sunil Karamchandani.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

remediation, and with the advent of artificial intelligence (AI), significant strides have been made in automating fault detection and analysis. Software engineers and quality assurance (QA) managers face numerous challenges in their respective positions. The following are often encountered challenges: evolving needs, intricacy of software systems, guaranteeing quality across many platforms and devices, upholding code quality, automating testing, addressing integration and compatibility problems, addressing security concerns, and managing limited resources.

In the last four decades, different researchers have worked on software defect identification, but their results have not been fruitful. In this regard, several ANN architectures have been proposed for software fault prediction and analysis by Begum and Dohi [1]. They presented a software fault prediction model that utilizes Box-Cox power transformation methods and compared existing software reliability growth models. A Multilayer Perceptron (MLP) is a sequentially linked artificial neural network with several layers of connections. In order to solve and foresee software errors, the authors of [1] developed an MLP with a one-stage look-ahead prediction utilizing Box-Cox Power Transformation as an experiment. Traditional ANN models' prediction performance was improved using a mix of ML and evolutionary computing approaches [2]. However, academics' main focus has shifted to the correct identification of a software defect, which has a significant impact on software release time. Several ANN methods for determining the best time to release software have been proposed by Begum and Dohi [3], [4]. Using a multi-stage look-ahead prediction approach where the ideal number of hidden neural networks and transformation values were discovered has limitations.

In another work, Liu et al. [5] investigated the impact of combining different sampling techniques and ML classifiers on defect prediction performance. While it finds no single optimal combination, it identifies support vector machines and deep learning as the most consistently performing classifiers. In [6], they contributed to the field of software defect prediction by utilizing various ML approaches to create multiple categorization or classification models, aiming to improve software quality and reduce testing costs. They also discussed the application of ensembling techniques and feature selection methods, such as principal component analysis (PCA), to further improve the accuracy of defect prediction models. They focused only on accuracy.

The authors [7] employed long-short-term memory (LSTM) networks in this kind of research to forecast software faults. They also calculated the data dispersion from the observed independent RMSE data points for each model. The quantified data dispersion value of the second model was found to be less minimal than the first one. The authors applied LSTM to predict the faults of multi-time stamps using a recursive approach. In addition, they compared LSTM and traditional software reliability growth models (SRGMs) based on their prediction accuracy evaluations.

The CNN model is well-suited for software fault prediction because it can effectively capture local and global patterns within source code structures, aiding in identifying potential defects. The author [8] improved a CNN model for within-project defect prediction (WPDP) and examined it with CNN and empirical results. This experiment uses 30-repetition holdout validation and 10×10 cross-validation. In WPDP experiments, the improved CNN model was equivalent to the existing CNN model and outperformed state-of-the-art machine learning models. One limitation of this research is the need for more data, specifically C/C++ open-source projects, to build robust and generalizable datasets for deep learning-based defect prediction.

Scope: This research introduces an LCNN for software defect prediction, which utilizes CNN models. The proposed LCNN model outperforms conventional ML models in terms of performance. Subsequently, we converted the executable strings into numerical values and included them in the CNN model, which comprises a word embedding layer, a pair of convolutional layers, two max-pooling layers, and one dropout layer. Then, we assessed the proposed model's prediction ability by examining its accuracy, MSE, and AUC using the CM1 dataset, taken from the NASA repository [9]. There were two CNN methods: 1D and 2D, which we used for this evaluation. 2D-CNN has almost the same outcomes compared with 1D-CNN. For root cause analysis, we use a method called "explainability," along with LIME and SHAP, to figure out what happened with a deep learning model that was trained on tabular data. On the other hand, 2D-CNN changes input data shape; for this reason, we are unable to use tabular data on XAI. Therefore, we use 1D-CNN for the recognition of the root causes of software faults. As a result, LIME is a good way to explain features of software defects in XAI, where we used it along with SHAP.

The following are some of our study's contributions:

- We have introduced an LCNN model that aims to enhance generalization and enable the identification of software defect features via the use of XAI.
- We used two CNN techniques: 1D and 2D, for experiments.
- We divided our research into two sections. In the first section, we use the CNN model for the prediction and choose 1D-CNN for further experiments. The subsequent experiment explained the underlying reasons for software issues.
- In our empirical discoveries, we carried out a hyperparameter search, during which we took into account the number of dense layers, the kernel size, and the stride step.
- LIME is a method that accurately approximates the predictions of any classifier or regressor using a locally interpretable model, allowing for truthful explanations.
- We introduce SHAP-based methodology, enhancing the interpretability of ML models and offering clearer insights into feature importance.

- Finally, we concluded that LIME and SHAP in XAI provide us with an accurate understanding of the root cause of software defects. However, when comparing them, we found that LIME gave better results than SHAP.

We introduce our research questions, which were answered in a section of analysis with LIME and SHAP.

- **RQ1: How can an LCNN architecture be designed for effective feature identification in software defect analysis using XAI?**
- **RQ2: What role does Explainable AI play in enhancing the transparency and interpretability of the LCNN model for software defect feature identification?**
- **RQ3: How does the LCNN architecture perform compared to traditional methods in terms of accuracy and efficiency for identifying software defect features?**
- **RQ4: What impact does the choice of hyperparameters have on the CNN model's performance in software defect feature identification, and how can these be optimized for better results?**
- **RQ5: How transferable is the proposed LCNN architecture to different software domains, and what factors influence its generalizability?**

The remaining parts of this paper are organized as follows: Section II presents a comprehensive review of previous research conducted in the areas of software defect identification and explainable AI. In Section III, we delve into the methodology, detailing the XAI techniques and feature engineering strategies employed. In addition, Section IV delivers the experimental results and examines the practical implications of this research. Finally, in Section V, we draw conclusions and outline future directions for this exciting intersection of software engineering and AI-driven interpretability.

II. RELATED WORK

The realm of software defect identification has seen a surge in research focusing on leveraging CNNs. In particular, the pursuit of developing lightweight and tailored CNN architectures has become a pivotal area of interest. Subsequently, it aims to enhance the efficiency and accuracy of defect identification within software systems. Integrating XAI techniques into this domain further augments the interpretability of these models, providing insights into the decision-making process of the network.

In addition, several researchers have delved into designing customized CNN architectures, considering the intricacies of software code while ensuring transparency in the identification process. Tong et al. [10] suggested a new way to solve the class imbalance problem with SDP that uses deep representations along with the two-stage ensemble and conducted an experiment on 12 NASA datasets. They have not worked on cross-project defect prediction, and there is no clear identification of root causes.

In [11], Zhu et al. introduced a new defect prediction model called DAECNN-JDP. This model utilizes a combination of denoising autoencoder and CNN techniques to provide just-in-time defect prediction. The evaluation of the model was conducted using six extensive open-source projects and compared to 11 baseline models. The experimental findings demonstrated that the suggested model surpasses these baseline models. However, they have not evaluated open-source and commercial projects. In addition, they have not used parameter optimization techniques to adjust the parameter. Subsequently, Qiu et al. [12] introduced a new method that utilizes a transfer CNN model to extract transferable semantic features for cross-project defect prediction (CPDP) tasks. The studies were carried out using 10 benchmark projects and 90 pairs of CPDP tasks.

Deep representation and ensemble learning were discussed in [13] for SDP to resolve the class imbalance problem. The experimental findings demonstrated that the proposed method outperformed existing cutting-edge techniques. In addition, to optimize defect prediction models, the authors in [14] proposed an ANN model with automated parameter tuning techniques. The results indicated that the performance of their proposed model improved after parameter settings were optimized. In [15], the authors improved the recurrent artificial neural network (RANN) method used to predict long-term software defects based on the number of software faults and proposed a simulation-based method (PI simulation) for calculating prediction intervals (PIs). In the end, they compared it to the conventional delta method in terms of the mean prediction interval width and PI coverage rate. Still, they have not validated software metrics, including McCabe, Halstead, and OO metrics.

Currently, the majority of SDP approaches have given little consideration to the expense associated with misclassifying faulty and non-faulty modules, with just a few instances where this has been considered [16], [17]. Nevertheless, the misclassification cost for the majority class is much lower compared to the minority class in the context of software testing. Cost-sensitive learning has shown its effectiveness in connecting various misclassification costs into the SDP process [18]. Faruk Arar and Ayan [16] tried to make cost-sensitive neural networks using cost-sensitive learning methods. They did this to try to fix the problem of the unequal distribution of classes by taking costs into account.

Zhao et al. [19] have introduced a new SDP model named Siamese parallel fully connected networks (SPFCNN), which combines the benefits of Siamese networks with DL. The experimental findings showed that the suggested model exhibits considerably superior performance compared to the benchmarked SDP techniques. In [20], a hybrid model that combines bidirectional long-short-term memory with CNN for SDP demonstrated the efficacy of the suggested methodology in accurately forecasting software problems. On the other hand, the authors introduced a Semantic Dependency Parsing (SDP) framework using an RNN that incorporates attention mechanisms [21].

TABLE 1. Contributions and limitations of different studies in the literature.

Ref.	Contributions	Limitations
[22]	They discussed the application of explainable artificial intelligence (XAI) in software bug classification.	They only used ML techniques in XAI.
[23]	The XAI methodologies were mostly used in the subfield of software engineering known as software fault prediction (SFP), and all investigations employed local explanations. These strategies have been used in combination with conventional machine-learning models.	The main causes of software defaults and the usage of R libraries for feature selection were not adequately addressed and analyzed to provide more precise and dependable results.
[24]	ML methods, including ANN, NB, and DTC, were used to generate software defects. The experiments included five different classes. Among the algorithms tested, NB and SVM have shown superior performance in identifying particular software flaws.	The examined study focused on the use of ML methods to forecast software defects. Nevertheless, they neglected to address the bugs responsible for their occurrence.
[25]	The researchers investigated SDP models by using a very efficient version of the XGBoost algorithm. This endeavor yielded millions of randomized models. After assessing the accuracy and interpretability of these models, it was determined that 3.5% of the total 1,997,287 models outperformed the seven traditional baseline models in the Jureczko datasets.	The provided sources do not explicitly mention any other biases or limitations of the information presented, and they used XAI techniques like LIME and SHAP for ML algorithms without discussing DNN procedures.
[26]	The researchers have used the concept of Explainable Artificial Intelligence (XAI) to enhance trust management while examining the Decision Tree (DT) model in the realm of cyber-security applications.	They encountered complications due to the absence of category variables with varying levels and datasets that were excessively overfitted. This problem enhances performance on the training dataset but exhibits suboptimal results on the test dataset.
[27]	They used LIME and SHAP techniques for an explanation of the root cause of software faults with a small number of features.	They did not use NN for evaluation; they performed only several ML techniques.

In Table 1, we show some contributions and limitations of different studies in the literature.

From the above-related work, we realized software defect prediction is vital for stakeholders, researchers, and the software industry. Therefore, we propose an LCNN for SDP using CNN models that outperform ML models. After that, we explain the root causes of the software defect by using expandable AI, and finally, we conclude that LIME in XAI outperformed compared with SHAP.

III. PROPOSED APPROACH

Fig. 1 depicts the five stages that make up our suggested approach architecture. The first phase consists of CM1 NASA datasets, preprocessing techniques, and splitting data for 1D-CNN and 2D-CNN. In the second phase, we experiment on 1D-CNN, and next, we experiment on 2D-CNN with process data. In the fourth step, we compared 1D-CNN and 2D-CNN to select the best CNN technique for software defect prediction. Finally, we have shown that LIME and SHAP are used to find the proper explanation and visualize the root cause of software defects.

A. FIRST PHASE

The following steps will describe the dataset used and the preprocessing methods.

1) DATASET

For this study, we used CM1 from the PROMISE database [9] for software engineering. It has 22 features that can be used to find defects in software, whereas the last one is a dependent feature, which indicates if there is a software defect or not. In addition, it has 6992 instances, where the class distribution is made up of a dataset with 3455 faults and 3537 no faults.

2) PREPROCESSING

In this research, we preprocess our datasets to transform raw data into a more presentable format. Here, we use the label encoding (LE) and standard scaler methods for normalization and balancing. The LE method [28] is used for measuring qualitative information. In this method, each category in a categorical variable is given an individual number that converts to an integer representation to make the data more accessible. Additionally, the standard scaler is used for scaling the data so that the distribution is zero with a standard deviation of one. The standard variation is as follows:

$$z_{\text{scaled}} = \frac{x - \mu}{\sigma} \tag{1}$$

where μ = Mean and σ = Standard Deviation.

Finally, we split the dataset by 80% for training and 20% for testing.

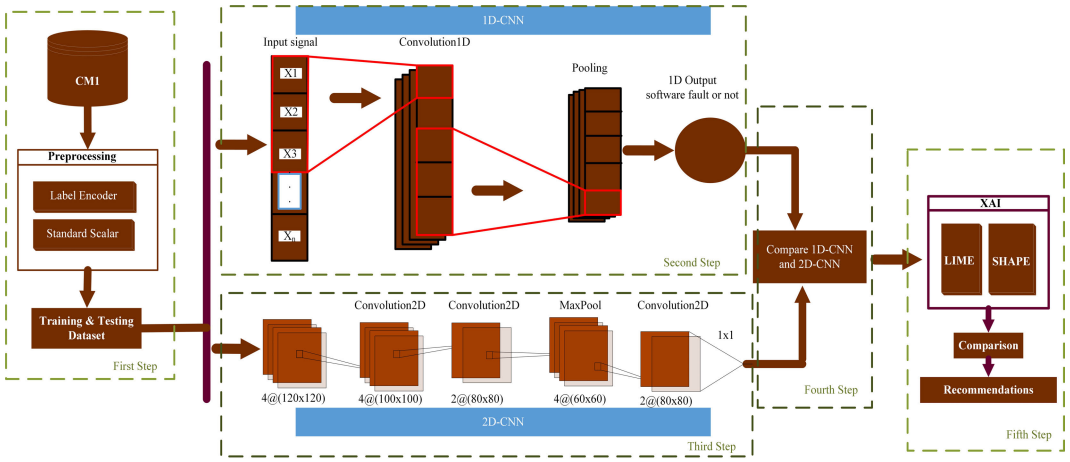


FIGURE 1. Comprehensive design of the proposed methodology.

B. SECOND PHASE (1D-CNN)

The one-dimensional convolutional NN (1D-CNN) is a special kind of ANN that works with sequential data by using convolutional operations to pull out features that are related to each other. It uses convolutional layers to automatically pull out hierarchical features from the input data. Additionally, it plays a crucial role in enhancing the accuracy and efficiency of predicting software defects by leveraging its ability to capture intricate dependencies within sequential code structures. Fig. 2 depicts the composition of a frequent 1D-CNN, which consists of three primary layers: The architecture of the 1D-CNN includes 1D convolutional layers, pooling layers, and fully connected layers [29]. Table 2 provides particular details on hyperparameters in the LCNN architecture for 1D-CNN. At first, the input data shape is (6992,1), with a kernel size of (3×1) and stride 1. For data compression, we used 3 times the convolution and Maxpolling techniques, and finally, we received the data shape (387,64). After that, we used the sigmoid activation function to classify software faults. There are two more important factors besides these three: the dropout layer and the activation function.

1) ONE-DIMENSIONAL CONVOLUTIONAL LAYER

The one-dimensional convolutional layer [30] applies convolutional operations along a single axis, extracting features and patterns from sequential data. The function takes the one-dimensional input (vector) $x[n]$ as its input, where n ranges from 0 to $N-1$. N is the total number of instances. The following parameters are used for making the layer.

1. **Kernels**: The kernels slide along the input sequence, capturing local patterns and producing output representations that highlight relevant features for further analysis. Let S represent the input sequence, K is the one-dimensional kernel and the resulting convolution output $\zeta[n]$ may be found by solving the (2):

$$\zeta[n] = (S * K)[i] = \sum_{k=0}^{K-1} S[i+k] \cdot K[k] \tag{2}$$

where $S * K$ denotes the convolution operation and i is the index of the output sequence.

2. **Activation function** [30]: An activation function in 1D-CNN introduces non-linearity, which is essential for learning complex patterns in sequential data. There are various types of functions, but the most popular are ReLU, Sigmoid, and Tanh, which are used for intricate relationships between features. We have used the exponential linear unit (ELU) as an activation function, which is a variant of the rectified linear unit (RELU). The ELU incorporates an additional alpha constant (α) to determine the smoothness of the function when the input values are negative. In addition, σ represents the input to the activation function and it exhibits a higher rate of convergence towards zero cost and yields more precise outcomes. The formula is expressed as $\alpha > 0$:

$$\text{ELU}(\sigma) = \begin{cases} \sigma & \text{if } \sigma > 0 \\ \alpha \cdot (\exp(\sigma) - 1) & \text{if } \sigma <= 0 \end{cases} \tag{3}$$

2) STRIDE [31]

Stride refers to the step size of a 1D-CNN. It determines the amount the kernel moves along the input signal. A larger stride reduces the output size by taking larger steps and extracting less information, while a smaller stride captures more detail but may increase computational complexity. Its adjustment allows CNNs to control the amount of information processed and influences the network's receptive field, impacting feature extraction in 1D sequences like time series or signals.

C. POOLING LAYER

The Pooling Layer [30] in a 1D-CNN condenses feature maps, reducing dimensionality and computational load. It is common to use Max Pooling, it selects the maximum value within a window, capturing the most prominent features. There are multiple forms of pooling procedures, including max pooling, sum pooling, and average pooling [31]. In the present investigation, we applied 1D max pooling, which

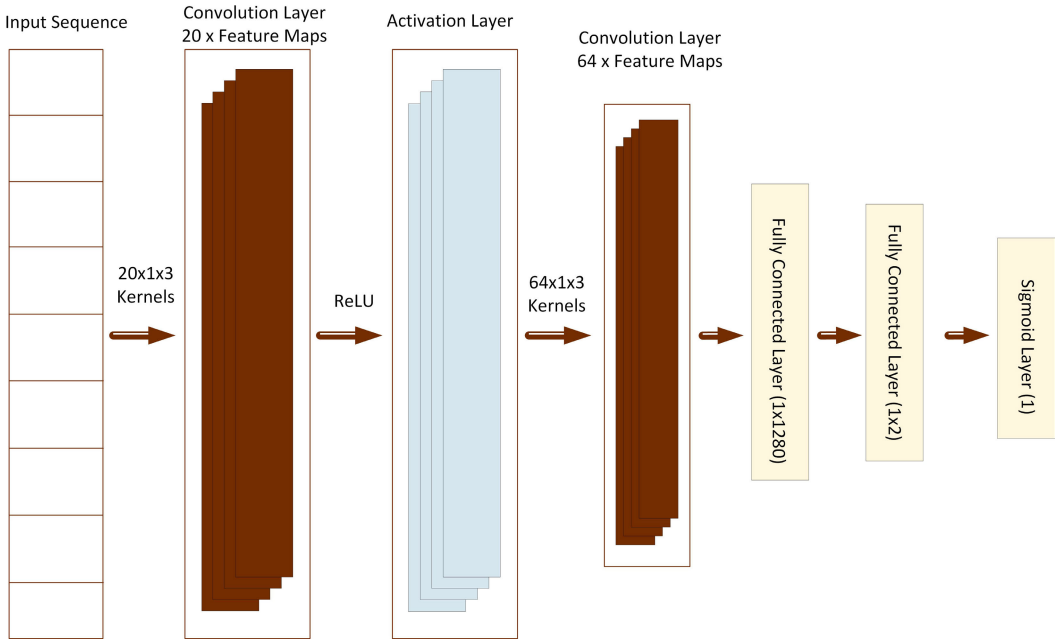


FIGURE 2. Architecture for the proposed basic 1D-CNN network.

entails analyzing the input data by applying a predefined pool size and stride and picking the maximum value from the examined area. In (4), the functioning of this can be illustrated by

$$\zeta_h^l = \max_{\forall p \in rh} \zeta_p^{l-1} \tag{4}$$

in which rh stands for the pooling area with the index h.

1) DROPOUT LAYER AND FLATTEN LAYER [30]

Dropout layers in 1D CNN randomly deactivate neurons during training, preventing overfitting by promoting robust learning. They improve model generalization by dropping a fraction of neurons, reducing interdependence. Flatten layers transform multidimensional arrays into a 1D array, essential in CNNs after convolutional layers, enabling seamless connection to fully connected layers. This process converts spatial information into a format suitable for traditional dense layers in the neural network. To fix this, we use a dropout layer, which removes some neurons from the neural network during training so that we end up with a smaller model.

2) FULLY CONNECTED LAYER [30]

The fully connected layer in a 1D CNN serves as the classifier, receiving flattened features from the convolutional layers. Each neuron in this layer connects to every output from the previous layer, aggregating high-level features to make predictions. It has dense connections that enable comprehensive feature extraction and pattern recognition. The activation function is a parameter of this function. The ELU function, defined in (4), is used in this study.

TABLE 2. Architecture of proposed 1D-CNN for tabular data.

Layer	Kernel size	Stride	Filters	Data shape
Input				(6992,1)
Conv1d_1	3 x 1	1	64	(6990,64)
Max_pooling1d_1	3 x 1	3 x 1		(2330,64)
Conv1d_2	3 x 1	1	64	(2328,64)
Max_pooling1d_2	3 x 1	3 x 1		(776,64)
Conv1d_3	3 x 1	1	64	(774,64)
Max_pooling1d_3	2 x 1	2 x 1		(387,64)
FC+BN				(193)
Activation (sigmoid)				(2)

D. THIRD PHASE (2D-CNN)

This research proposes leveraging 2D-CNN for software defect prediction using tabular data. Traditional methods face limitations in extracting intricate patterns from tabular datasets. Our approach aims to preprocess the tabular data into 2D representations, enabling the application of CNNs. The methodology involves data transformation into 2D matrices, utilizing CNN layers for feature extraction, and integrating predictive models. The study's focus is on improving accuracy and efficiency in software defect prediction. Table 3 provides particular details on the LCNN architecture hyperparameters for 2D-CNN. The batch size, usually denoted by "None," is the first dimension of the output data shape. The resulting feature map's spatial dimensions follow: It is a 4 × 5 grid with 32 channels and 160 parameters. None, 2, 2, 32 max_pooling2d params 0: Max-pooling layer

TABLE 3. Architecture of proposed 2D-CNN for tabular data.

Layer	Output Shape	Param #
conv2d	(None, 4, 5, 32)	160
conv2d_1	(None, 4, 5, 32)	4128
max_pooling2d	(None, 2, 2, 32)	0
conv2d_2	(None, 2, 2, 64)	18496
conv2d_3	(None, 2, 2, 64)	36928
max_pooling2d_1	(None, 1, 1, 64)	0
flatten	(None, 64)	0
dense	(None, 128)	8320
dense_1	(None, 1)	1
Activation (sigmoid)	(None, 1)	1

applied to preceding convolutional layer output. Reduces spatial dimensions by 2, and each of the 32 feature maps is treated separately. No parameters are trainable in this layer (0, output shape: None, 2, 2, 32). flatten (None, 64) This flattening layer turns the preceding layer's 3D output into a 1D vector. The output shape is (None, 64), and this layer has no trainable parameters (0). The output layer we have used is activated using the sigmoid function.

Some methodology we used for 2D-CNN:

- Transformation of tabular data into a 2D format akin to an image-like structure, representing relationships between software metrics.
- Normalization and feature engineering to enhance the network's ability to discern patterns.
- Designing a 2D CNN architecture with convolutional and pooling layers to capture local and global feature dependencies.
- Incorporating multiple convolutional layers to learn hierarchical representations from the tabular data
- Splitting the dataset into training, validation, and testing sets.
- Training the 2D CNN model on the transformed tabular data, adjusting hyperparameters for optimal performance.
- Validating the model's performance using various evaluation metrics like precision, recall, F1 score, and accuracy.
- Visualization techniques to interpret the learned features and understand the significance of various software metrics in defect prediction.
- Analyzing the performance metrics to highlight the superiority and efficiency of the 2D CNN for software defect prediction.

E. FOURTH PHASE

In the context of comparing the performance of 1D-CNN and 2D-CNN for software defect prediction, many metrics have been used such as accuracy, AUC, and MSE [32], [33].

1) ACCURACY

Accuracy is a fundamental metric for evaluating the overall performance of the proposed LCNN architecture. It is defined as the ratio of correctly predicted instances to the total number of instances in the dataset. Higher accuracy refers to the model's capacity to accurately categorize instances as either faulty or non-defective. It is computed as follows:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Number of Predictions}} \times 100\% \quad (5)$$

2) MEAN SQUARED ERROR (MSE)

MSE is a statistic for calculating the average squared difference between what was expected and what happened. A smaller MSE in software defect predictions indicates more accurate and reliable results, highlighting improved precision in forecasting potential defects within the software system.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \quad (6)$$

where n is the number of instances, y_i is the actual value of the i-th instance, and $\hat{y}_i$ is the predicted value of the i-th instance.

3) AREA UNDER THE CURVE (AUC)

The AUC is a key measure of how well the model can tell the difference between defective and non-defective cases at different choice levels. In addition, the ROC shows how the true positive rate and false positive rate change as limits are raised or lowered. In combinally, the AUC takes the ROC curve and turns it into a single number from 0 to 1, and a higher AUC means better discrimination.

F. FIFTH PHASE (XAI)

A new study method is being suggested to look into how XAI techniques can be used to find and understand the causes of software errors. By using XAI the fields of software engineering and artificial intelligence with useful information that could be used to make software more reliable and easier to maintain. Therefore, we have focused on creating and using a brand-new XAI-based model to analyze software bugs. We used advanced XAI methods, like LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive Explanations), to give clear and understandable information about how complicated software systems work.

1) LIME

LIME (Local Interpretable Model-agnostic Explanations) plays a crucial role in advancing software defect identification by providing transparent and interpretable insights into the decision-making process of complex ML models. This methodology empowers researchers and practitioners to enhance model trustworthiness and pinpoint potential vulnerabilities, contributing to more effective and reliable software defect detection strategies. LIME offers several benefits for software fault root cause analysis, such as interpretability, local explanations, model agnostic, *and so on*. This investigation holds promise for advancing the field of software engineering by providing a nuanced understanding of the root causes behind software faults through the lens of LIME's interpretability.

2) PSEDOCODE FOR LIME

Utilizing a 1D-CNN model, this research integrates LIME for interpretable predictions, extracting feature importance to pinpoint software fault root causes. In Algorithm 1, we demonstrated how to build a model using the suggested LIME in XAI methods. The method empowers transparency in complex models, aiding effective root cause analysis in software fault detection. First, it shows the faults as input-output pairs. Next, it starts up a LIME explainer, writes LIME explanations for each fault, looks at the explanations to find the root causes, and finally shows out the root causes.

Algorithm 1 LIME Pseudocode for Finding the Root Cause of Software Faults

Input: Software fault data, and fault manifestations
Output: Root causes of software faults
`Step1:` Represent software faults as input-output pairs

```
fault_data = []
for each instance fault in faults do
    fault_data.append((fault.code,
fault.execution_environment, fault.manifestation))
end for
```

`Step2:` Initialize LIME explainer

```
lime_explainer = LimeExplainer(kernel_width=0.3,
class_weights=0: 1, 1: 1)
```

`Step3:` Generate LIME explanations for individual faults

```
explanations = []
for fault_input, fault_output in fault_data do
    explanation = lime_explainer.explain_instance(fault_input,
fault_output, labels=[0, 1])
    explanations.append(explanation)
end for
```

`Step4:` Analyze LIME explanations to identify root causes

```
root_causes = []
for explanation in explanations do
    root_cause = []
    for feature, weight in explanation.as_list() do
        if weight > 0.1 then
            root_cause.append(feature)
            root_causes.append(root_cause)
        end if
    end for
end for
```

`Step5:` Output root causes of software faults

```
return root_causes
```

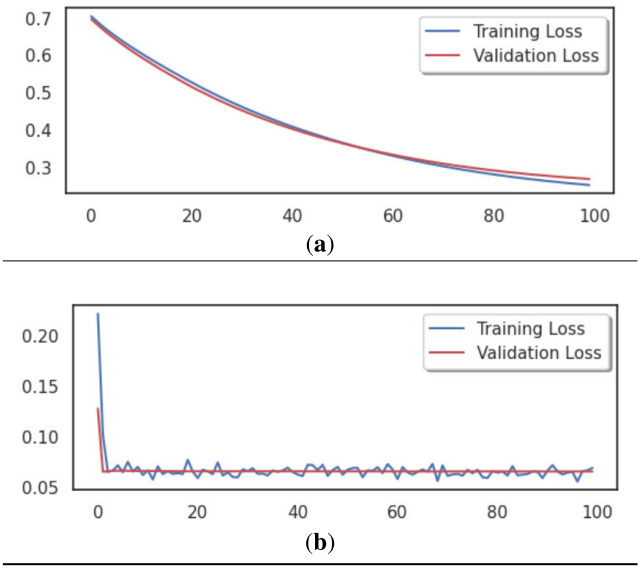
3) SHAP

SHAP (SHapley Additive exPlanations) plays a pivotal role in advancing software defect identification by providing a robust framework for interpreting and understanding the contributions of different features in predictive models. This technique enables a nuanced examination of the impact of individual features on the prediction of software defects, fostering transparency and interoperability. The integration of SHAP in defect identification models enhances their explainability, contributing to more informed and effective decision-making in software development and quality assurance processes. It offers several benefits for software fault root cause analysis, such as contribution quantification, global and local explanations, model agnostic, and so on.

TABLE 4. Performance evaluation for 1D-CNN and 2D-CNN.

Technique	Accuracy	MSE	AUC
1D-CNN	0.9145	0.0855	0.5000
2D-CNN	0.9281	0.0648	0.9292

TABLE 5. Plot the loss and accuracy curves for training and validation. (a) 1D-CNN, (b) 2D-CNN.

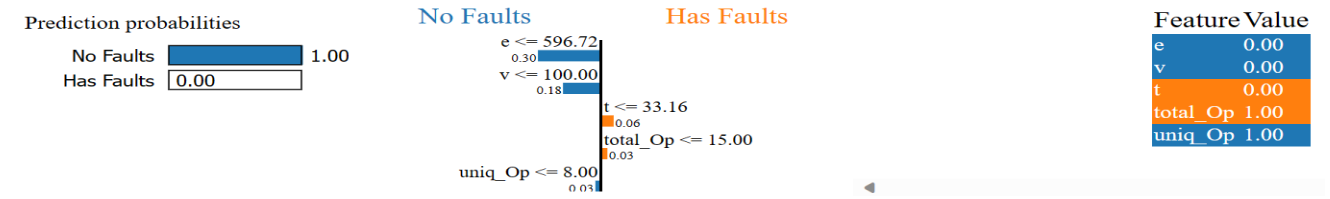


4) PSEDOCODE FOR SHAP

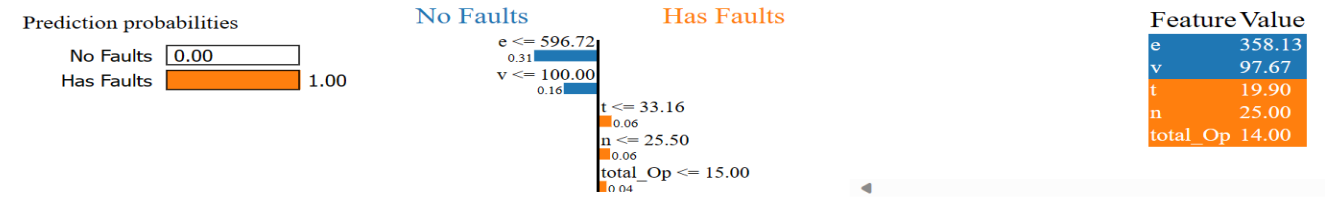
Utilizing pseudocode for SHAP in our research elucidates the interpretability of software defect identification models, offering a concise and clear representation of the Shapley values' computation for enhanced understanding and application. In Algorithm 2, we illustrated the process of constructing models utilizing XAI techniques for the recommended SHAP.

The function identify_root_causes accepts as input a trained machine learning model, a collection of code snippets, and their related test cases. The generate_SHAP_explanations function is invoked to compute the SHAP values for each code snippet. SHAP values quantify the individual impact of each feature on the model's prediction for a specific code snippet. The get_top_features method is used to determine the most prominent features for each code snippet. The following characteristics have the greatest SHAP values, signifying their substantial impact on the model's fault prediction. The analyze_top_features function is utilized to scrutinize the most prominent attributes of each code snippet. This involves examining the interaction between the qualities, their roles within the code, and their potential impact on the system's behavior. A list is

TABLE 6. LIME agnostic explanations prediction. here, no faults by (a) and faults by (b).



(a)



(b)

Algorithm 2 SHAP Pseudocode for Finding the Root Cause of Software Faults

Input: model, code_snippets, test_cases
Output: Root causes of software faults

Calculate SHAP values for each code snippet
root_causes = []
Iterate through each code snippet and its corresponding SHAP values
fault_data = []
for *code_snippet*, *SHAP_value* in zip(*code_snippets*, *SHAP_values*) **do**
    Extract the top features based on SHAP values
    top_features = *get_top_features* (SHAP_value)
    Analyze the top features to identify root causes
    root_causes_for_snippet = *analyze_top_features* (code_snippet, top_features)
    Append the identified root causes for the current snippet
    root_causes.extend(root_causes_for_snippet)
end for
//Return the list of identified root causes
return root_causes

utilized to maintain the identified underlying reasons for each code snippet. The function identify_root_causes generates an exhaustive compilation of all the identified root causes for each code snippet.

IV. RESULT ANALYSIS

The primary objective of our study was to design and evaluate a lightweight customized CNN architecture for the identification of software defect features. We investigated it using Python code for each dataset individually, as well as the

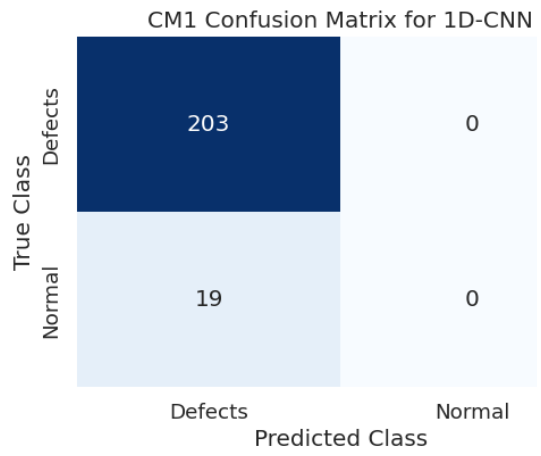


FIGURE 3. Confusion matrix without function that is sensitive to cost.

1D-CNN and 2D-CNN methods. Here discussed the results we got for each of the datasets.

A. MODEL ACCURACY AND EFFICIENCY

Both 1D-CNN and 2D-CNN models demonstrated commendable accuracy rates in identifying software defect features. In Table 4, 1D-CNN exhibited an accuracy of 91.45%, while the 2D-CNN achieved a slightly higher accuracy of 92.81%. On the other hand, MSE for both models were robust, indicating their ability to correctly identify positive instances with minimal false positives. Furthermore, in Table 5 we showed the loss and accuracy curves for both the 1D-CNN and 2D-CNN techniques. The AUC values for both architectures were noteworthy, emphasizing their ability to capture a high percentage of actual positive instances. A comparative analysis with both CNN techniques showcased we have chosen 1D-CNN for further explanation by using expandable AI because 1D-CNN data shape is the same before and after the training, and testing. However, when examining the confusion matrix shown in Fig. 3, it is evident

TABLE 7. SHapley additive exPlanations prediction of software faults.

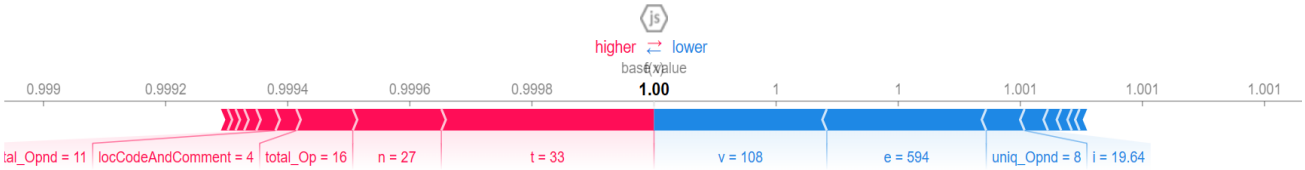


TABLE 8. Research question and answer based on our research.

SL.	Question	Answer
RQ1	How can an LCNN architecture be designed for effective feature identification in software defect analysis using XAI?	The design of a lightweight customized CNN for software defect analysis involves modifying the architecture to balance computational efficiency and model interpretability. Employing XAI techniques, such as LIME or SHAP, helps elucidate CNN's decisions. Features like reduced depth, convolutional layers, and global pooling enhance efficiency while facilitating transparent insights into software defect identification. The resulting model combines accuracy with interpretability, addressing the intricate nature of software defect analysis.
RQ2	What role does Explainable AI play in enhancing the transparency and interpretability of the LCNN model for software defect feature identification?	Explainable AI techniques, such as LIME or SHAP, will be incorporated to elucidate the decision-making process of the CNN model. By generating human-interpretable explanations for individual predictions, stakeholders gain insights into the features influencing defect identification. This transparency fosters trust in the model's output, making it more accessible and usable for software engineers and developers.
RQ3	How does the LCNN architecture perform compared to traditional methods in terms of accuracy and efficiency for identifying software defect features?	The performance of the LCNN architecture will be rigorously evaluated against traditional methods, considering metrics like accuracy, precision, recall, and computational efficiency. Comparative analysis will provide insights into the model's effectiveness in accurately identifying defect features while maintaining a lightweight structure suitable for real-world applications.
RQ4	What impact does the choice of hyperparameters have on the CNN model's performance in software defect feature identification, and how can these be optimized for better results?	Hyperparameter tuning will be conducted to investigate the sensitivity of the CNN model's performance to parameters such as learning rate, batch size, and kernel size. Through systematic optimization, the study aims to identify the optimal set of hyperparameters that maximizes the model's accuracy and efficiency in identifying software defect features.
RQ5	How transferable is the proposed LCNN architecture to different software domains, and what factors influence its generalizability?	Generalizability is crucial for the practical applicability of the proposed LCNN architecture. The research will explore the transferability of the model across various software domains by assessing its performance on diverse datasets. Factors such as dataset characteristics, class imbalances, and domain-specific features will be analyzed to understand the model's adaptability and identify potential limitations.

that the accuracy of defect prediction has shown a substantial improvement. Specifically, for the CM1 datasets using 1D-CNN, the prediction accuracy has increased from 91% to 9%.

B. EXPLAINABILITY AND INTERPRETABILITY

In the context of software defect prediction using the proposed LCNN architecture, LIME, and SHAP can be employed to identify the most influential features for each defect prediction.

1) VISUALIZATION OF LIME

LIME explanations revealed that the LCNN architecture consistently identified relevant features associated with software defects. In Table 6, we show the top five most effective features out of twenty-one features for an explanation of the root cause of software defects. Table 6.(a) suggests that there will be no faults. The *e* (30%) total features ratio and *v* features had the most significant effect on the model's ability to estimate. It was found that the *t* and *total_OP* features are the best for a software fault to happen. We conclude that these two features

have become very important for finding defects in software when using LIME. Additionally, Table 6.(b) illustrates the important features used to analyze the software defects in the dataset. The characteristics *t*, *n*, and *total_Op* of the software faults in this dataset have experienced a favorable influence.

2) VISUALIZATION OF SHAP

SHAP explanations provided further insights into the relative importance of different features for each defect prediction. The base value in Table 7 is laid at 1.00, which is referred to as a prediction value. Red-colored features have a positive impact, causing the predicted value to go closer to 0. If the *t* feature is removed, the forecast will decrease from 0.9897 to 1.00. In contrast, characteristics that are colored blue have a negative impact, meaning they pull the forecast value closer to 1. Removing the feature of *v* will increase the prediction rate from 0.90 to 1.

3) ANALYSIS OF LIME AND SHAP

Both LIME and SHAP are valuable explainability techniques that provide insights into the decision-making process of the LCNN architecture. However, they differ in their approach and provide complementary information. We demonstrated the visualization of LIME and SHAP to identify the root causes of software defects. However, For the comparison, LIME shows better than SHAP because LIME provides a clear concept of root causes. After that, we easily find unfavorable features that are the main cause of software defects. In Table 8, we have discussed the research question and answer for explaining our methods.

V. CONCLUSION

As part of this study, we started making a lightweight CNN design to identify the root causes of software defect features. To make software better, it is important to know why bugs occur in the testing phase. In this research, we have tried to make an advanced CNN model for finding the reason for software bugs quickly and easily. The dataset went through a lot of preparation to make sure it would work with CNNs. We addressed issues such as label encoding, standard scaller, and reshaping to meet the input requirements of both 1D-CNN and 2D-CNN models. In addition, we experimented with both 1D-CNN and 2D-CNN models to evaluate their performance in identifying software defect features. Then these models were trained and tested on relevant datasets to assess their effectiveness. Subsequently, we evaluated the 1D-CNN architecture due to its superior performance in capturing spatial relationships within defect features, aligning well with the nature of software defect identification. To enhance interpretability, we employed LIME and SHAP techniques specifically for the 1D-CNN model. These explainability tools provided valuable insights into feature importance, aiding in understanding the decision-making process of the model in the domain of software defect identification.

VI. STRENGTHS, LIMITATIONS, AND FUTURE PERSPECTIVES

Our research has exhibited several strengths. The LCNN architecture showcases efficiency in identifying software defects, providing a lightweight solution for practical deployment. By integrating XAI, particularly the SHAP method, the interpretability of the model is improved, thereby promoting confidence and comprehension in the decision-making process. Using the inclusion of the PC1 Promise Repository dataset increases its practical applicability, hence strengthening the strength of this research.

Despite its strengths, the research has limitations. The effectiveness of the proposed solution may be context-dependent, and its generalizability to diverse software environments needs validation. The proposed approach provides reliance on a specific dataset, however in practical applications may limit the model's adaptability to various software development practices.

Future research could focus on expanding the model's applicability by testing it on a broader range of datasets. On the other hand, LCNN can collaborate with industry practitioners to provide valuable insights to address specific software development challenges as well as ensure its effectiveness.

ACKNOWLEDGMENT

(Momotaz Begum, Mehedi Hasan Shuvo, and Imran Asharaf contributed equally to this work.)

REFERENCES

[1] M. Begum and T. Dohi, "A neuro-based software fault prediction with box-cox power transformation," *J. Softw. Eng. Appl.*, vol. 10, no. 3, pp. 288–309, 2017, doi: 10.4236/jsea.2017.103017.

[2] S. Noekhah, A. A. Hozhabri, and H. S. Rizi, "Software reliability prediction model based on ICA algorithm and MLP neural network," in *Proc. 7th Int. Conf. e-Commerce Developing Countries*, Apr. 2013, pp. 1–15.

[3] M. Begum and T. Dohi, "Optimal release time estimation of software system using box-cox transformation and neural network," *Int. J. Math., Eng. Manage. Sci.*, vol. 3, no. 2, pp. 177–194, Jun. 2018.

[4] M. Begum and T. Dohi, "Optimal stopping time of software system test via artificial neural network with fault count data," *J. Quality Maintenance Eng.*, vol. 24, no. 1, pp. 22–36, Mar. 2018. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/JQME-12-2016-0082/full/html

[5] Y. Liu, W. Zhang, G. Qin, and J. Zhao, "A comparative study on the effect of data imbalance on software defect prediction," *Proc. Comput. Sci.*, vol. 214, pp. 1603–1616, Jan. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922020610

[6] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software defect prediction analysis using machine learning techniques," *Sustainability*, vol. 15, no. 6, p. 5517, Mar. 2023. [Online]. Available: https://www.mdpi.com/2071-1050/15/6/5517

[7] M. R. Islam, M. Begum, and M. N. Akhtar, "Recursive approach for multiple step-ahead software fault prediction through long short-term memory (LSTM)," *J. Discrete Math. Sci. Cryptography*, vol. 25, no. 7, pp. 2129–2138, Oct. 2022. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/09720529.2022.2133251

[8] C. Pan, M. Lu, B. Xu, and H. Gao, "An improved CNN model for within-project software defect prediction," *Appl. Sci.*, vol. 9, no. 10, p. 2138, May 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/10/2138

[9] J. S. Shirabad and T. Menzies, "The PROMISE repository of software engineering databases," School Inf. Technol. Eng., Univ. Ottawa, Ottawa, ON, Canada, Tech. Rep., 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff

[10] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Inf. Softw. Technol.*, vol. 96, pp. 94–111, Apr. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917300113

[11] K. Zhu, N. Zhang, S. Ying, and D. Zhu, "Within-project and cross-project just-in-time defect prediction based on denoising autoencoder and convolutional neural network," *IET Softw.*, vol. 14, no. 3, pp. 185–195, Jun. 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.2019.0278

[12] S. Qiu, H. Xu, J. Deng, S. Jiang, and L. Lu, "Transfer convolutional neural network for cross-project defect prediction," *Appl. Sci.*, vol. 9, no. 13, p. 2660, Jun. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/13/2660

[13] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques," *Expert Syst. Appl.*, vol. 144, Apr. 2020, Art. no. 113085. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419308024

[14] Z. Yang and H. Qian, "Automated parameter tuning of artificial neural networks for software defect prediction," in *Proc. 2nd Int. Conf. Adv. Image Process.* New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 203–209, doi: 10.1145/3239576.3239622.

[15] M. Begum, M. S. Hafiz, M. J. Islam, and M. J. Hossain, "Long-term software fault prediction with robust prediction interval analysis via refined artificial neural network (RANN) approach," *Eng. Lett.*, vol. 29, p. 44, Aug. 2021.

[16] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Appl. Soft Comput.*, vol. 33, pp. 263–277, Aug. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494615002720

[17] P. Kumudha and R. Venkatesan, "Cost-sensitive radial basis function neural network classifier for software defect prediction," *Sci. World J.*, vol. 2016, pp. 1–20, Sep. 2016, doi: 10.1155/2016/2401496.

[18] S. Viaene and G. Dedene, "Cost-sensitive learning and decision making revisited," *Eur. J. Oper. Res.*, vol. 166, no. 1, pp. 212–220, Oct. 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221704002978

[19] L. Zhao, Z. Shang, L. Zhao, T. Zhang, and Y. Y. Tang, "Software defect prediction via cost-sensitive Siamese parallel fully-connected neural networks," *Neurocomputing*, vol. 352, pp. 64–74, Aug. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231219305004

[20] A. B. Farid, E. M. Fathy, A. S. Eldin, and L. A. Abd-Elmegid, "Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM)," *PeerJ Comput. Sci.*, vol. 7, p. e739, Nov. 2021.

[21] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Sci. Program.*, vol. 2019, pp. 1–14, Apr. 2019, doi: 10.1155/2019/6230953.

[22] Ł. Chmielowski, M. Kucharzak, and R. Burduk, "Application of explainable artificial intelligence in software bug classification," *Informatyka, Automatyka, Pomiary Gospodarce Ochronie Środowiska*, vol. 13, no. 1, pp. 14–17, Mar. 2023, doi: 10.35784/iapgos.3396.

[23] A. Khan, A. Ali, J. Khan, F. Ullah, and M. A. Khan, "A systematic literature review of explainable artificial intelligence (XAI) in software engineering (SE)," *Researchsquare*, pp. 1–28, Sep. 2023. [Online]. Available: https://www.researchsquare.com/article/rs-3209115/v1

[24] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software bug prediction using machine learning approach," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 2, pp. 1–6, 2018, doi: 10.14569/ijacsa.2018.090212.

[25] G. dos Santos, E. Figueiredo, A. Veloso, M. Viggiato, and N. Ziviani, *Predicting Software Defects With Explainable Machine Learning*. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 1–10.

[26] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (XAI) to enhance trust management in intrusion detection systems using decision tree model," *Complexity*, vol. 2021, pp. 1–11, Jan. 2021, doi: 10.1155/2021/6634811.

[27] M. Begum, M. H. Shuvo, I. Ashraf, A. A. Mamun, J. Uddin, and M. A. Samad, "Software defects identification: Results using machine learning and explainable artificial intelligence techniques," *IEEE Access*, vol. 11, pp. 132750–132765, 2023.

[28] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *J. Big Data*, vol. 7, no. 1, p. 28, Apr. 2020, doi: 10.1186/s40537-020-00305-w.

[29] E. Chaerun Nisa and Y.-D. Kuan, "Comparative assessment to predict and forecast water-cooled chiller power consumption using machine learning and deep learning algorithms," *Sustainability*, vol. 13, no. 2, p. 744, Jan. 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/2/744

[30] M. Saini, U. Satija, and M. D. Upadhayay, "One-dimensional convolutional neural network architecture for classification of mental tasks from electroencephalogram," *Biomed. Signal Process. Control*, vol. 74, Apr. 2022, Art. no. 103494, doi: 10.10162Fj.bspc.2022.103494.

[31] J. Rala Cordeiro, A. Raimundo, O. Postolache, and P. Sebastião, "Neural architecture search for 1D CNNs—Different approaches tests and measurements," *Sensors*, vol. 21, no. 23, p. 7990, Nov. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7990

[32] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 297–308, doi: 10.1145/2884781.2884804.

[33] K. Wongpheng and P. Visutsak, "Software defect prediction using convolutional neural network," in *Proc. 35th Int. Tech. Conf. Circuits/Systems, Comput. Commun. (ITC-CSCC)*, Jul. 2020, pp. 240–243.

MOMOTAZ BEGUM received the M.Sc. degree in CSE from DUET, Gazipur, Bangladesh, in 2013, and the Ph.D. degree in information engineering from Hiroshima University, Japan. Her Ph.D. study was fully funded by Japanese Government, specifically the Ministry of Education, Culture, Sports, Science and Technology [Monbukagakusho (MEXT)], from October 2014 to September 2017. She is currently a Distinguished Professor with the Department of Computer Science and Engineering, DUET. Her specialization lies in software rejuvenation, software aging, and human pose recognition. She has published several journals and conference papers in the world's reputed journals and conferences on the different topics of computer science, such as the IoT, machine learning, and neural networks. Her research interests include software reliability, software engineering, artificial neural networks, data mining, data clustering, information systems, and analysis.

MEHEDI HASAN SHUVO (Member, IEEE) is currently pursuing the B.Sc. degree in engineering with the Department of Computer Science and Engineering, Dhaka University of Engineering & Technology, Gazipur, Bangladesh. He is a very courteous and dedicated person. He possesses strong technical skills and he loves challenging work to create engaging and informative content. He has three years of industrial job experience in the field of mobile application development (Android and iOS). He worked full-time in the three professor's laboratories at his university, which improved him. His research interests include federated learning, explainable artificial intelligence, computer vision, the IoT, machine learning, deep learning, object detection, and water and environmental sustainability. He is an ACM Member.

MOSTOFA KAMAL NASIR received the B.Sc. degree in computer science and engineering from Jahangirnagar University, Dhaka, Bangladesh, in 2000, and the Ph.D. degree in mobile ad-hoc technology from the University of Malaya, Kuala Lumpur, Malaysia, in 2016. He is currently a Professor of computer science and engineering with Mawlana Bhashani Science & Technology University, Tangail, Bangladesh. His current research interests include VANET, the IoT, SDN, and WSN.

AMRAN HOSSAIN (Member, IEEE) was born in Chandpur, Bangladesh, in 1981. He received the B.Sc. and M.Sc. degrees (Hons.) from the Department of Computer Science and Engineering, Dhaka University of Engineering & Technology, Gazipur, Bangladesh, in 2009 and 2015, respectively, and the Ph.D. degree in the field of microwave brain imaging (biomedical imaging) from Universiti Kebangsaan Malaysia (UKM), Malaysia, in 2022. He is currently an Associate Professor with the Department of Computer Science and Engineering, Dhaka University of Engineering & Technology. Before this, he joined as a Lecturer with Dhaka University of Engineering & Technology, in 2012, where he was an Assistant Professor, in 2015. He is the author of 15 research journal articles, four conference papers, and a few book chapters on various topics related to computer programming language, information system analysis and design, and computer graphics. His research interests include wireless communication antenna design, microwave brain imaging and biomedical imaging using deep learning, cognitive radio networks (CRN), software reliability and quality assurance, wireless communication with machine learning, image processing, and segmentation using deep learning. He is a member of the Institute of Engineers, Bangladesh (IEB).

MOHAMMAD JAKIR HOSSAIN (Member, IEEE) was born in Cumilla, Bangladesh, in 1981. He received the B.Sc. and M.Sc. degrees (Hons.) from the Department of Electrical and Electronic Engineering, Dhaka University of Engineering & Technology, Gazipur, Bangladesh, in 2006 and 2013, respectively, and the Ph.D. degree in the field of metamaterial applications (metamaterial-based absorber) from Universiti Kebangsaan Malaysia (UKM), Malaysia, in 2019. He has been a Professor with the Department of Electrical and Electronic Engineering, Dhaka University of Engineering & Technology, since 2020. He has authored or coauthored approximately 26 refereed journals, five book chapters, and six conference papers. His research interests include metamaterials, metamaterial-based absorbers, antennas, the IoT, wireless communication, microwaves, and satellite communication. He is a member of the Institute of Engineers, Bangladesh (IEB). He was awarded the First Prize and the Best Paper at the International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE), in 2022.

IMRAN ASHRAF received the M.S. degree (Hons.) in computer science from Blekinge Institute of Technology, Karlskrona, Sweden, in 2010, and the Ph.D. degree in information and communication engineering from Yeungnam University, Gyeongsan, South Korea, in 2018. He was a Postdoctoral Fellow with Yeungnam University, where he is currently an Assistant Professor with the Information and Communication Engineering Department. His research interests include positioning using next-generation networks, communication in 5G and beyond, location-based services in wireless communication, smart sensors (LIDAR) for smart cars, and data analytics.

JIA UDDIN received the M.Sc. degree in telecommunications from Blekinge Institute of Technology, Sweden, in 2010, and the Ph.D. degree in computer engineering from the University of Ulsan, South Korea, in January 2015. He is currently an Assistant Professor with the AI and Big Data Department, Endicott College, Woosong University, South Korea. He is also an Associate Professor (currently on leave) with the CSE Department, BRAC University, Bangladesh. His research interests include fault diagnosis using AI and audio and image processing. He was a member of the Self-Assessment Team (SAC) of CSE, BRACU, in the HEQEP project funded by the World Bank and the University of Grant Commission Bangladesh, from 2016 to 2017.

MD. ABDUS SAMAD (Member, IEEE) received the Ph.D. degree in information and communication engineering from Chosun University, Gwangju, South Korea. He was an Assistant Professor with the Department of Electronics and Telecommunication Engineering, International Islamic University Chittagong, Chattogram, Bangladesh, from 2013 to 2017. He has been a Research Professor with the Department of Information and Communication Engineering, Yeungnam University, South Korea. His research interests include signal processing, antenna design, electromagnetic wave propagation, applications of artificial neural networks, deep learning, and millimeter-wave propagation by interference and/or atmospheric causes for 5G and beyond wireless networks. He won the prestigious Korean Government Scholarship for his doctoral study.