SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I6P118

Volume 12 Issue 6, 227-237, June 2025

Original Article

Hybrid Deep Learning Model for Software Defect Classification Using Code2Vec and LinkNet-BiLSTM Score Level Fusion

Srinivasa Rao Katragadda[1], Sirisha Potluri[2]

[1,2]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Bowrampet, Hyderabad, Telangana, India.*

[1]*Corresponding Author : ksrresearch1@gmail.com*

Received: 10 April 2025 Revised: 11 May 2025 Accepted: 12 June 2025 Published: 27 June 2025

Abstract *- The software package faults stance an important trial, impacting systems' reliability, functionality, and security. Traditional methods for defect detection, including manual inspections and static tools, are often insufficient for handling large, complex codebases. Current progress in ML and DL provides more robust solutions by identifying complex patterns within data. It presents a hybrid model that combines Code2Vec for feature extraction, Improved LinkNet for spatial data processing, and Bi-LSTM for sequential pattern analysis, leveraging score-level fusion to improve software defect classification. Code2Vec transforms unstructured source code into dense vector representations, capturing critical semantic and syntactic features. Improved LinkNet excels in extracting high-level structural features, while Bi-LSTM captures long-term dependencies in code sequences. The proposed score-level fusion integrates the outputs of these models to harness their complementary strengths, reducing noise sensitivity and enhancing accuracy. The fused scores are passed to a soft-max classifier to foresee a given snippet code that is fault-motionless or non-defect-prone. The final output classifies the software defect into specific categories, if applicable, based on the trained dataset. It demonstrates that the hybrid model outperforms metrics, accuracy, precision, recall, and F1 scores, existing methods in package fault prophecy. The study also highlights the significance of hyperparameter tuning and training large, labelled datasets. This research contributes to scalable, efficient defect detection methods that address real-world challenges in software development, setting a foundation for future improvements in predictive analytics and automated software quality assurance. The proposed model, implemented in Python, enhances classification performance, achieving an accuracy of 92%.*

Keywords *- Software defect classification, Deep learning, Code2Vec, Bi-LSTM, Score-level fusion.*

1. Introduction

Software defects are a significant issue for software development processes across the software development life cycle, affecting the software systems' reliability, functionality, and sometimes security. Detecting the software defects at earlier stages of the development life cycle has been imperative for reducing the cost of bug fixes while at the same time enhancing software quality. The prior methods used traditionally include visual examination of the code, manual inspections, and static tools and rules [1]. However, these methods fail to exercise large and complex codebases and most of the time, they need active human intervention, which makes them time-consuming and error-pragmatic. In the past decade, ML techniques and, more recently, DL techniques have increasingly applied in software defect prediction context due to their capability of 'learning' and identifying complex and quite 'hidden' patterns in a large number of datasets that would commonly aid in automating and very much improving the defect inspection process. A recent development in the field is to train deep learning models that will be able to learn the underlying structure of the unstructured data present in the form of source code efficiently without much influence of manual feature extraction [2]. One such research could be the improvement of multiple deep learning paradigms and implementing them in a hybrid manner so that the best of every architecture is harnessed. These models can be especially effective when used in software defect prediction because it may be the subtle architecture within these systems that identifies areas prone to defects. This research study aims to combine Code2Vec, Improved LinkNet and Bi-LSTM in score-level fusion to propose a probable solution to the defect forecast problem that is accurate, effective and scalable. This paper establishes that the feature extraction stage is crucial for properly functioning machine learning algorithms for defect detection [3]. Source code is highly unstructured, and therefore, it is usually

complicated to find meaningful features from relations that could be sparse, complex, and non-linear. It is addressed by Code2Vec, a new method designed for producing vectors for code. With Abstract Syntax Trees (ASTs), another feature extraction tool named Code2Vec is capable of parsing out semantic and syntactic features inherent in code. This method converts code into comprehensible vectors regarding the code's components and their work, making it optimal for defect prediction tasks [4]. The model translates paths in the AST into sequences of elements in code constructs such as loops, variables, and functions and then aggregates the sequences into a vector of fixed dimensionality for each source code fragment. These vectors retain more information from the code. The information flow from encoded semantic and syntactic levels can be fed into the classification models [5]. However, the primary problem of attempting to analyse the data once the features have been extracted using Code2Vec is imperfect whether or not to classify. On this behalf, extracting features through Improved LinkNet and Bi-LSTM presents improved features for processing and classification [6]. An extension of the network called Improved LinkNet, which is effective when dealing with complex data representations, has also been applied in this work to classify the software defects and to capture the spatial dependencies of the features extracted from the source code. Usually, a Recurrent Neural Network (RNN) is more appropriate for sequential, Bidirectional-Long Short-Term Memory (Bi-LSTM) as it handles long-term dependencies and contextual information. In software packages, it enables the model to capture that the code is sequential, where the order of statements, functions or variables may form part of the problem in detecting defects [7].

However, applying only one deep learning model for the defect classification is sometimes problematic because of the model bias and general capabilities for signal feature extraction from the input. This is where our theory of score-level fusion comes into play. In score-level fusion, the results of Improved LinkNet and Bi-LSTM models are combined at the score level to produce a final output. The idea is that one or another model can complement the advantages of the other model in terms of more minor variance and better pattern identification. For instance, although Improved LinkNet may perform better in obtaining the high-level features linked to the structural characteristics of the code, BI-LSTM may perform better in sequences and temporal characteristics of the code. Combining multiple models through integration, the hybrid system provides more accuracy and less noise sensitivity. This research discusses identifying software as defective as the focal concern in the data analysis process of Code2Vec. Feature extraction is done using Improved LinkNet and Bi-LSTM for classification. The enhanced LinkNet performs well for the elaborate formation of data representation. Bi-LSTM deals with Long-term and contextual information. The score-level fusion theory fuses Improved LinkNet and Bi-LSTM models on the score level to adopt both advantages while reducing variation and enhancing pattern recognition. This system has less noise sensitivity and better accuracy than the old one. The fusion process often combines the outputs of one or more models, for example, probabilities or classes, by simple or complex methods [8]. The proposed hybrid deep learning model at the score level fusion minimises the errors, enhances the generality, and augments the efficiency of multiple models for defect classification. It is, therefore, compared with current methods for demonstrating improvements brought by score-level fusion and the hybrid deep Learning model. To train a combined model, highlight that it requires a large dataset of code annotated with defects labels. These are fed into Improved LinkNet and Bi-LSTM models before being subjected to the score-level fusion strategy. Hyperparameter neural networks based on size, layers, and reading rate, which are so important, will be selected using feature selection methods. The research concern of this paper is to improve software defect detection through the integration of Code2Vec for feature representation and Improved LinkNet and Bi-LSTM for classification outcomes [9].

Key points are discussed as below:

- Code2Vec transforms unstructured source code into dense vector representations, effectively capturing semantic and syntactic features for accurate defect identification.
- A modified LinkNet architecture extracts high-level spatial features, ensuring robust analysis of code structures for defect classification.
- Bi-LSTM captures that enhancing the understanding of temporal and contextual relationships.
- Score-level fusion combines outputs from LinkNet and Bi-LSTM, leveraging their complementary strengths to reduce noise and boost classification performance.
- The model detects defect-prone code and categorizes defects into specific types based on the dataset, aiding deeper insights.
- Achieving 92% accuracy, the model outperforms traditional methods, demonstrating scalability and suitability for large, complex codebases.

It is organized as follows: The relevant work is given in section 2. The problem statement is defined in section 3. The suggested approach is explained in section 4. The experimental results are obtainable in Section 5. The conclusion and future work are delivered in Section 6.

2. Related Works

Borandag [10] focuses on the growing importance of retaining machine learning and deep learning. It adopts the SFP XP-TDD dataset from three software projects and compares five classifiers and their hybrids. The research discovers that DL algorithms perform better than ML models when dealing with large sets and include RNN-based models

tested using Eclipse and Apache Active MQ datasets. The obtained results demonstrate the significance of DL in the SFP in terms of the sample size of the datasets. However, it has certain drawbacks, such as the restriction of the analysis to specific datasets, limited appropriateness of the proposed method for comparatively small datasets, and issues concerning the overall generality of the results to real-life realistic composite software applications.

Batool and Khan [11] examine SFP through DL techniques, including LSTM, BiLSTM, and RBFN, with datasets derived from Chidamber and Kemerer (CK) features. As can be inferred, both LSTM and BiLSTM perform better than the traditional models, whereas the RBFN model has been seen to be faster in computation. Moreover, k-fold authentication is used, and the accuracy of the models is better than that of the other available accuracy models. The paper emphasizes that DL can improve SFP mechanisms while revealing weaknesses, such as reliance on data sets, difficulty extending the results to different environments, and high computational overhead related to training effective DL models. [12] recommends using CNN and MLP merging to develop a DL approach for predicting software faults based on twelve datasets from the PROMISE repository. Carrying out class imbalance with SMOTEND, the lowest accuracy, 0.195 for MLP, is established, surpassing CNN. Some of the significant advantages of the model include the following: First, it is relatively simple; second, it is flexible, as is demonstrated by its capability to accommodate input data. There are several limitations: synthetic oversampling techniques, the possibility of overfitting with relatively small samples, and no direct comparison between the proposed architectures and other DL structures aside from MLP and CNN.

Das et al. [13] present the PM2-CNN model that combines the transformer architecture with the multi-channel CNN for SFP. It integrates defect-related data that consists of natural language inputs, such as comments within the code and commit messages with the code's programming for a richer representation. The results show a positive shift in generic evaluation metrics. Constraints include computational processing needs due to transformer elements, difficulties combining different input types, and dependence on pre-trained language models, which may be suboptimal for all scenarios.

Rathi et al. [14] assesses SFP employing six feature selection procedures, nine sampling methods, and six classifiers over 56 OSPs. Concluding the class imbalance and redundancy, the present study identifies that SMOTEE with the correlation-based Feature Selection (FS2) delivers the best results. Using these techniques, it records enhanced AUC of more than 75 per cent of projects' post implementing the techniques. Lack of constraints includes the large number of evaluations per project equal to 792 dataset combinations, the risk of overfitting due to high parameter tuning, and difficulties applying these methods to SFP pipelines.

Yang et al. [15]. present a weighted association rule-based algorithm for SFP with an emphasis on the degree of contribution of features to generate satisfactory rules and enhance the prediction rate. Thus, by handling class imbalance and improving rule generation and ranking, the proposed model significantly acquires the average F1 score improvement of about 6.4% and the average MCC improvement of about 9.8% compared to the existing approach on the PROMISE dataset. Still, there are specific problems, such as increased computational time for large datasets, difficulty in generating rules, and finally, the applicability of this rule-based system is not easy with the changing trends in the software development paradigm.

3. Problem Statement

The proposed model struggles to generalize across diverse datasets and environments. While DL models, such as RNN-based architectures, CNN, and MLP k-fold cross-validation, have shown superior performance in handling large datasets, they face challenges related to high computational overhead, reliance on specific data characteristics, and difficulties in extending results to real-world, composite software applications. Additionally, approaches incorporating hybrid models or feature selection techniques often contend with issues like class imbalance, overfitting, and computational inefficiencies. Notwithstanding, it is developing SFP methods to efficiently handle diverse, real-life scenarios while minimizing the need for extensive data preprocessing and feature tuning. To address these challenges, this research proposes "Score-Level Fusion in Hybrid Deep Learning with Extractive Feature Sets for Software Defect Classification." This approach leverages score-level fusion to combine and identify defects in diverse software environments. By integrating different learning models and optimizing feature selection, this method aims to improve generalization, reduce overfitting, and provide more efficient, scalable solutions for real-world SFP tasks.

4. Hybrid Code2Vec-Improved LinkNet-BiLSTM Score Level Fusion Model

A Hybrid Deep Learning Model for software defect classification uses advanced techniques to boost accuracy. It first involves data gathering and processing where missing values are resolved, and features of the codes are extracted from Code2Vec, thus converting snippets of code into their corresponding vector forms, using the extracted feature for two architectures models that are classification in LinkNet modified for this classification. In contrast, bidirectional input in time was provided for the Bi-LSTM architecture. Finally, a score-level fusion combines the outputs from both models to achieve better classification accuracy by leveraging the strength of each approach. This structured methodology guarantees robust defect identification, as shown in Figure 1.

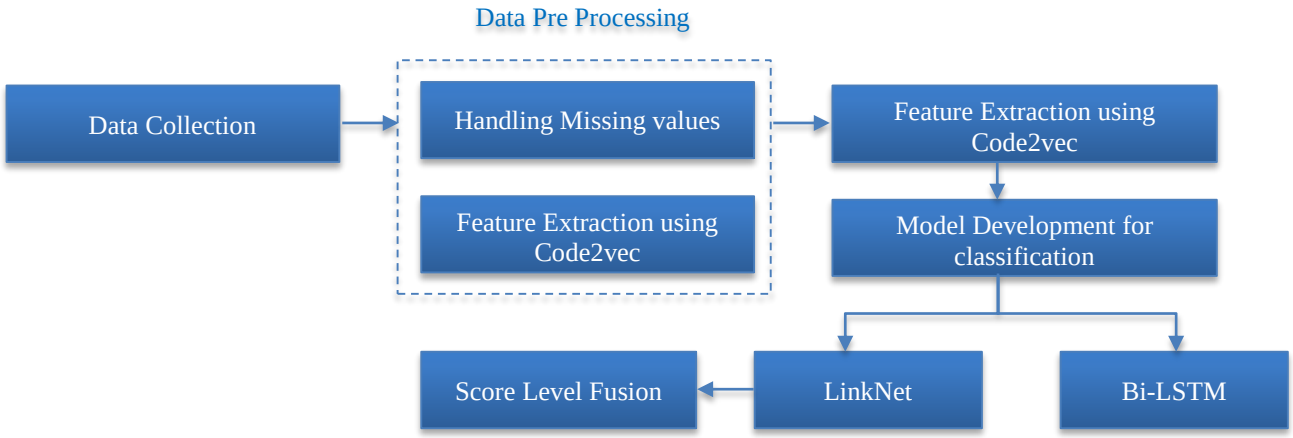

Fig. 1 Block diagram of proposed methodology for enhancing robotic performance

4.1. Data Collection

The data used in this work is drawn from the PROMISE repository, a commonly used software project repository containing data on software project defects. Each dataset contains project names, names of the source files with defects, versions, number of defects and 20 traditional features. These features characterize the software metrics frequently used to predict a defect. To match prior work and replicate their findings, source code was downloaded. There are two project models, the Within-Project Defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP), with one project as the source project and the other as the target project. Due to this setup, it is possible to comprehensively evaluate the defect prediction models across various conditions [16].

4.2. Data Preprocessing

Data preprocessing is a significant process utilized to clean and set up the data before feeding it for model development.

4.2.1. Handling Missing Values

Missing values are detrimental to machine learning techniques, as they may hinder the software's performance in providing important sources. As for it, numerical features with missing values can be filled with mean or median to keep the magnitude beneficial without taking much bias. In the case of categorical data, their mean can be used in its place. In ordinal data, the median Reply can be used instead of it. If missing data is disproportionately high in a feature, then that feature can be completely trimmed off. It was done to guarantee that the data set would not be altered in the process of modeling and that key information is preserved.

4.2.2. Feature Normalization and Scaling

Feature extraction in Code2Vec is complex and leverages the conversion of source code into dense vectors. Using this methodology on datasets like the one presented here with projects like Camel, Lucene, and Log4j, the methodology can ideally pick out important features crucial to defect prediction and other uses of software analysis. It uses the abstract syntax tree and path-oriented representations to identify the logic used in code.

Step 1: Parsing Source Code into AS

The feature extraction starts with parsing the Abstract Syntax Tree (AST). It connects to the tree of the language, for example, a variable declaration, a method call or a control structure. Another feature of this tree structure is the identification of relations such as parent-child relations of loops and conditions or inheritance.

For example, the AST separated initialization, condition, and increment statements in the for-loop case as different yet related elements. In the given dataset, each file of the projects, for example, Lucene or Xalan, would be transformed to the corresponding AST. This process means that the structural elements expressed in the extracted features include relations like either depth of inheritance or the coupling between objects to reflect the traditional metrics as captured.

Step 2: Path-Based Representations

After the construction of the AST, the process of analysis in Code2Vec yields path-based representations, which are the core Kent features. A path in the AST implies one specific path between two nodes and describes the system. For example, a path may link a variable declaration site to the site of the method call, showing how data is processed in the program.

Regarding the position of projects in the buggy rate, it is reasonable to expect that, for example, projects on POI or Lucene exhibit specific defect-related patterns in their pathways. Some routes could highlight where logical mistakes can happen, such as when checking invalid pointers, like the null pointer, or where a loop ought to be and whether it exists. Code2Vec cleans up the relationship database, ensuring it only contains pertinent data, and eliminates these paths to stop noisy components from entering as input.

Step 3: Token Embedding's

The obtained AST path is then converted to a series of dense vector embeddings per each node and edge in this path. Like Word2Vec in NLP, it uses the neural network-based embedding technique to map tokens such as the variable names, operators, and keywords to a continual vector space. Such embeddings are designed to capture similarity, so working elements (such as "sum" and "total") would be nearer.

For instance, in the dataset files, code smell patterns are such things as unused variables or inadequate exception management, which likely have shared embeddings between projects. These are then quantified, thanks to Code2Vec, and the buggy or non-buggy behavior patterns across different projects can be generalized.

Step 4: Aggregating Path Embeddings

Code2Vec summarizes all the path embedding related to a given file into a compact representation of the fixed dimensionality. This vector gives structural and contextual information on all the paths in the AST of the code snippet, which sums up the snippet. There are methods, including the attention mechanism, which help to increase the weight of critical paths so that necessary features, such as high cyclomatic complexity or deep nesting, are considered. For the appearance of bugs, this step would group strings of paths often associated with bugs, such as deep recursion in Xalan tightly coupled classes in Log4j, among others. These aggregated embeddings are used as inputs to the defect prediction models and hence identify some features that are not always included in the features, such as LOC or WMC.

Due to the features extracted from Code2Vec described above, one can solve problems when analyzing the given dataset. It can easily surpass antiquated qualitative metrics such as inheritance depth or cohesion by learning domain patterns unseen by traditional methods. This is particularly important in high buggy rate scenarios, which often require recognizing pesky but utterly important bugs like misuse of library functions or improper usage of inheritance. Therefore, the feature extraction of Code2Vec transforms large code patterns into simpler and semantically dense features. Using AST paths, token embeddings, and aggregation techniques gives a strong and self-organizing method for dissecting the code patterns, which improves the defect prediction in any dataset with different buggy rates and project complexities.

4.3. Feature Extraction Using Code2Vec

Feature extraction in Code2Vec is complex and leverages the conversion of source code into dense vectors. Using this methodology on datasets like the one presented here with projects like Camel, Lucene, and Log4j, the methodology can ideally pick out important features crucial to defect prediction and other uses of software analysis. It uses the abstract syntax tree and path-oriented representations to identify the logic used in code.

4.4. Classification Using LinkNet for Software Defect Detection

LinkNet architecture takes data related to software defects through its encoder and decoder segments, which unlearn and relink the feature maps to maximize classification. They added several convolutional layers to enhance light and fit the real-time defect classification tasks. Improved LinkNet's structure incorporates elements present in ResNet, with additional residual connections to enhance performance. The Improved LinkNet34 architecture is employed for software defect detection as described in the method flowchart. Out of all the blocks in the model, the primary block commences by performing convolution of input feature maps using a kernel of sizes 7×7 times 7 and a fixed stride of 2. After this, maximum pooling again decreases the number of parameters accepted as the input for the intensity of the subsequent process. After that, a set of residual blocks with small outputs relearn the representation of input features. During the downsampling of the input, the first convolutional layer of each residual block uses a stride of 2 for efficient feature downsampling. The rest of the layers work with a stride of 1 to preserve features in the spatial domain. Architecture also embodies a sequence of decoder modules, each connected to an encoder module. The decoder starts with a 1 × 11 times 1 convolution layer to down ample individual feature maps and ends with batch normalization and a convolution transformation that up samples feature maps. This design helps sustain the restoration of features and lowers computational load [17].

In LinkNet architecture, dilated convolutions expand the receptive field without expanding the parameters. The variable dr controls the spacing between the kernel weights, which affects the sampling density of the feature map. Let us consider the kernel in Equation (1)

$$ks = ks + (ks + 1) \times (dr - 1) = dr \times (ks - 1) + 1 \quad (1)$$

Where represents the original kernel size. The dilation rate is defined dr. For a given layer, the receptive field size$r$ f_n can be expressed in Equation (2)

$$rf_n = x_{n-1} \times rf_{n-1} + ks_{n-1} - x_{n-1} \quad (2)$$

Here, denotes the stride at layer n and ks_{n-1} is the effective kernel size of the dilated convolution.

The main advantage of Improved LinkNet pertains to the ability to integrate the encoder and decoder modules. The decoding architectures of the prior art may discard valuable spatial data as successive decoding occurs. To overcome this,

Improved LinkNet uses pooling indices as non-training parameters to connect the encoder and decoder to maintain spatial relationships. During oversampling, this connection bypasses the encoder's input directly to the decoder's output and is useful in reconstructing lost spatial details. Furthermore, by making this direct connection from the encoder to the decoder, the architecture scales down the parameters, thus making it very efficient for real-time analysis of software defect classification. This design is superior to prior arts because it provides a good trade-off between precision and complexity, and the practical application of solving large-scale defect datasets is feasible.

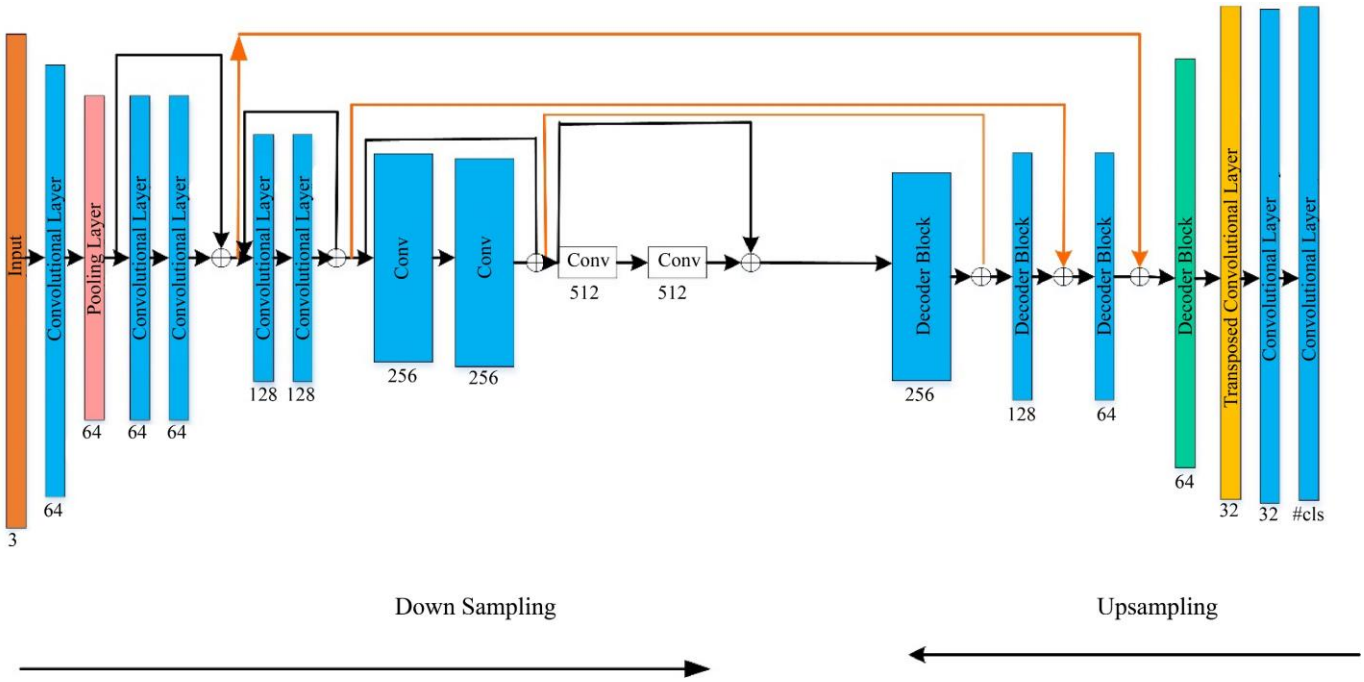

Fig. 2 Architecture of LinkNet

4.5. Classification using Bi LSTM

In mathematical notations, let us first introduce the forward and backward computations of the Bi-LSTM network within a sequence modeling context, like software defect prediction. Bi-LSTM sequential data is more appropriate for problems that require understanding deep temporal features such as defect detection of code. Below are the key components and equations describing its operation:

4.5.1. Forget Gate

This forgets that. f_t lies between 0 and 1, of h_{t-1} and x_t with 0 signifying complete loss of information and 1, a complete recall of information.-1) this is given in Equation (3)

$$f_{t=}\sigma(W_{fh}[h_{t-1}], M_{fa}[a_t],b_f) \quad (3)$$

4.5.2. Input Gate

It has two parts: a sigmoid layer $i_t$that determines the importance and a tanh layer c_t that generates new candidate values. The Equations are (4), (5)

$$i_t= \sigma(w_{ih}[h_{t-1}], u_{ix}[x_t],b_i) \quad (4)$$

$$c_t=\tanh (w_{ch}[h_{t-1}], u_{cx}[x_t] \quad (5)$$

While c_t LSTM memory is filled with a vector of fresh candidate values. i_t Indicates whether the value should be improved or not. Its numerical formula is shown in Equation (6) in the following formula.

$$c_t=t_f*c_{t-1}+i_t*c_t \quad (6)$$

While t_f determines the result of forgetting the gate number within zero, and one represents the final value that was attained; one represents the final value that was retained

4.5.3. Exit Gate

A sigmoid function o_t determines the importance, and the final hidden state h_t is obtained by applying a $tanh$ function to the updated cell state c_t this is mathematically shown in Equation (7)(8)

$$o_t=\sigma(w_{oh}[h_{t-1}], M_{ox}[x_t]b_o) \quad (7)$$

$$h_t = o_t * \tanh (c_t) \quad (8)$$

Where w_o, u_o, b_o are the weights and biases for the output gate.

A Bi-LSTM involves using both forward and backward LSTMs to capture both direction's dependencies. The forward hidden state h_t Moreover, Equation (9) offers the final output:

$$h_t=[h_t; h_t] \quad (9)$$

The defect prediction on the combined output h_t Passed to a fully connected layer and softmax activation, Equation (10) provides a method for calculating the probabilities of each class.

$$y = softmax(W_y h_t+b_y) \quad (10)$$

Here: W_y and b_y are weights and biases for the output layer. Using the bidirectional nature of Bi-LSTM, the network learns both the preceding and succeeding context, which is important in tasks such as software defect prediction, where errors may depend on long-range relationships in the sequence [18]. BILSTM Architecture is presented in Figure 3.

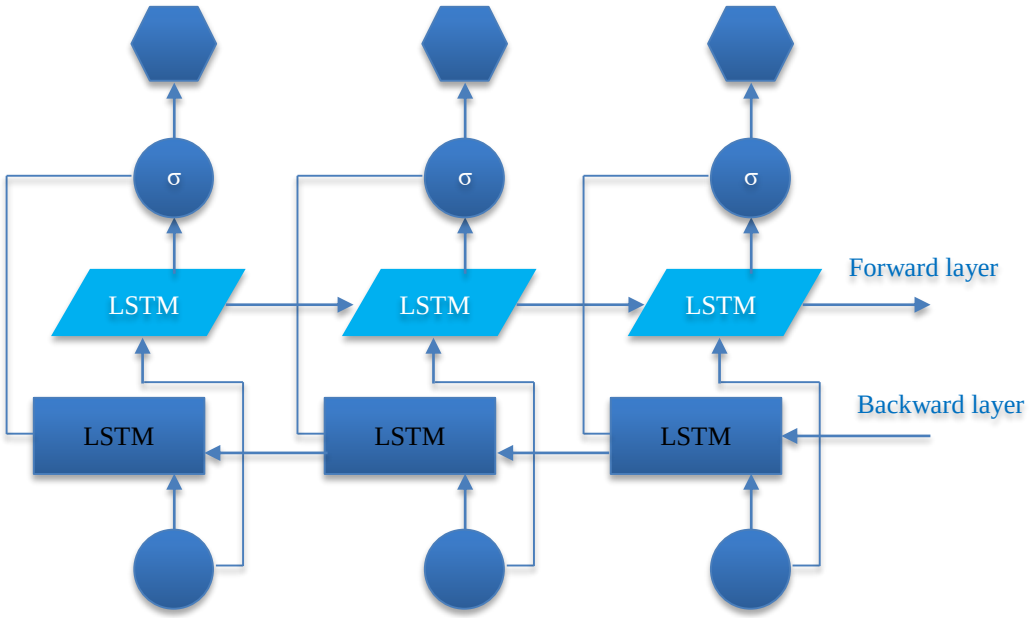

Fig. 3 BILSTM Architecture

4.6. Score-Level Fusion

The score-level fusion procedure is employed to mix the results of several models to provide a more precise and reliable prediction result. This work employs the point scores calculated based on Improved LinkNet and Bi-LSTM models that indicate the possibility of a defect being contained in the corresponding software. The concept of fusion is intended to combine these scores in a way that takes advantage of the individual capabilities of each model in the general decision-making process. Below is a detailed explanation of the fusion strategies:

4.6.1. Weighted Averaging

With weighted averaging, scores from the two models are mixed by providing different weights to the output of each model depending on their value or accuracy. This method ensures that the model with better performance or higher confidence in its decision makes a bigger value contribution to the final decision in Equation (11)

$$S_{final} = w_1 . S_{linknet} + w_2 . S_{Bi-LSTM} \quad (11)$$

Where: S_{final} is the fused score. $S_{linknet}$ And $S_{Bi-LSTM}$ are the scores from Improved LinkNet and Bi-LSTM, respectively.
w_1 *and* w_2 are weights such that w_1+w_2

4.6.2. Learned Fusion Models

For all the fusion strategies mentioned in this paper, learned fusion models are preferred due to their versatility and capacity to fuse Improved LinkNet and Bi-LSTM more effectively. Here, a meta-model relying on the evaluation results for the two models is built using other models, such as neural networks or logistic regression, to define the most suitable pair for defect prediction.
The learned fusion approach incorporates data characteristics during the fusion process, and it is highly stable even in large, complex and noisy data. It can also include other functionalities or model outputs if required since it is scalable for the new conditions. Despite more computational complexity and data demands needed for implementation, this method produces algorithms with better precision and robustness than basic methods such as weighted mean or maximum fusion.

5. Result and Discussion

Accurateness increased from 85% to 92% after hyperparameter tuning, highlighting its potential for precise defect prediction. Training time analysis showed that the hybrid model required approximately 200 seconds, compared to 120 seconds for Improved LinkNet and 150 seconds for Bi-LSTM, reflecting the trade-off between efficiency and accuracy. The confusion matrix revealed that the model correctly classified most defective and non-defective components, with minor misclassifications. These findings confirm the effectiveness of score-level fusion in combining the strengths of multiple models for enhanced software defect detection.

5.1. Training and Testing Accuracy of ARIMA - DRL

Figure 4 represents training and validation accuracy over 30 epochs. The training accuracy was high for comparison. The blue trend shows training accuracy; however, this curve rises straight with epochs, almost attaining a value of 92, meaning it can better classify defects by epochs. The orange line represents validation accuracy, which increases initially but then starts to plateau and oscillates at approximately 0.9 after around 10 epochs. This behavior means the model performs better with the validation set but has a saturation later and might even begin to stabilize - perhaps because the model is failing to generalize the things it is being fed. This plot plots the performance curves of both training and validation sets and shows some overfitting effects. The other software offers projects for defect classification.

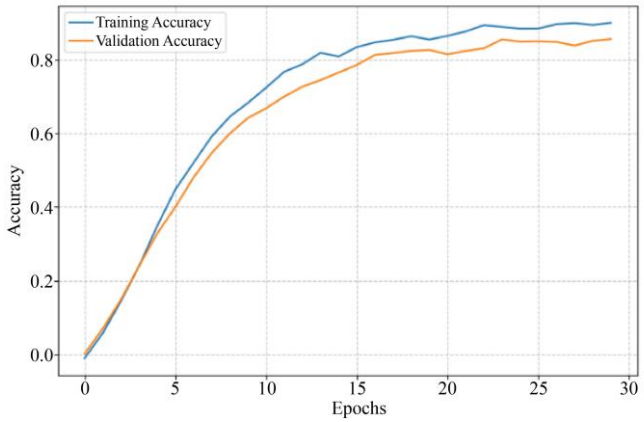

Fig. 4 Accuracy over Epoch

5.2. Training and Testing Loss of ARIMA -DRL

Figure 5 describes training and validation loss plots used to classify software defects. It increases and approaches unity with epochs on the horizontal axis as the model improves its ability to classify defects in the training set. The orange line indicates the validation accuracy, which initially improves but stabilizes to around 0.9 and oscillates after 10 training epochs. This behavior indicates that although set in the beginning, it starts stalling, which indicates that the model might be having problems generalizing new unseen data. This graph shows the accuracy of the model and validation data sets and indicates that the model is overfitted. As the previous sections pointed out, training accuracy increased while validation accuracy remained stagnant, which implied that the model might overfit the data and not work well on other software projects for defect classification.

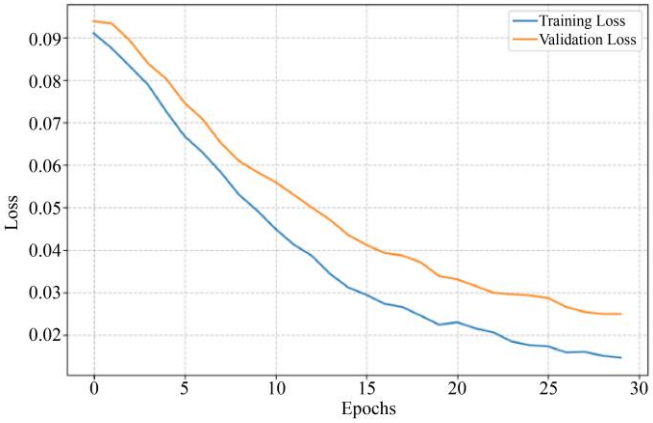

Fig. 5 Loss over Epoch

5.3. Accuracy before and Hyper parameter Tuning

Figure 6 explains that The labels on the horizontal axis are “Before Tuning” and “After Tuning,” while the accuracy scale on the y-axis ranges from 0.800 to 1.000. The findings acquired before and after tuning are displayed in bar charts with values of 0.85 and 0.92. Understanding this pronounced improvement in accuracy demonstrates that hyperparameter adjustment is still essential for bettering the chosen model’s performance and improving its capacity to anticipate software flaws.

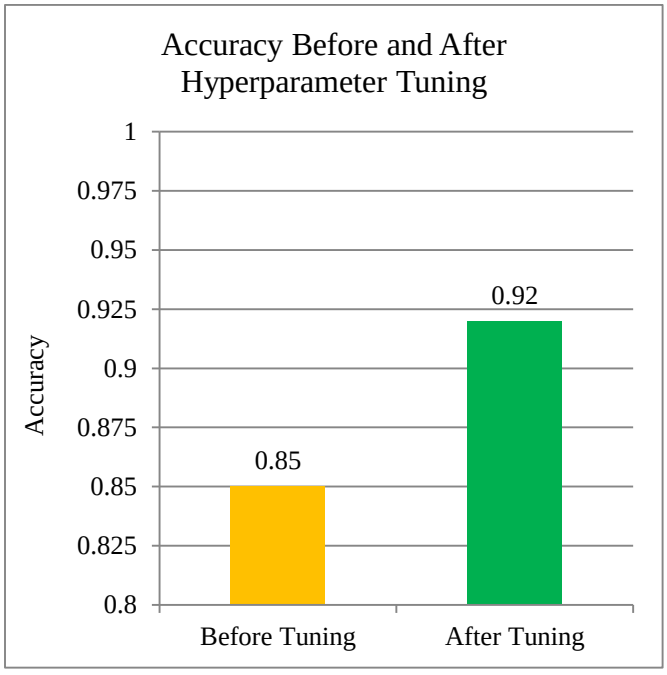

Fig. 6 Hyper parameter tuning graph

5.4. Training Time Comparison

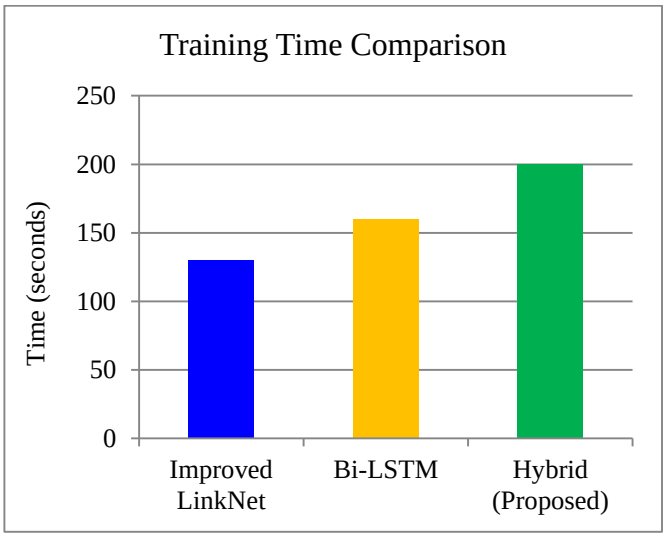

Fig. 7 Model training time comparison

Figure 7 equivalences, which are Improved LinkNet, Bi-LSTM, and the Hybrid (Proposed), are shown for predicting software defects. On the x-axis, there are the models, and on the y-axis, the training time is in seconds. The above graph shows that the training time needed by the model Improved LinkNet is approximately 120 seconds, Bi-LSTM is approximately 150 seconds, while the Hybrid (Proposed) took the highest time of approximately 200 seconds. This comparison illuminates the trade-off between accuracy and efficiency, where while the hybrid model promises potentially better prediction accuracy, it takes the maximum training time. Hence, it is an essential choice regarding the practical application of software defect prediction in software.

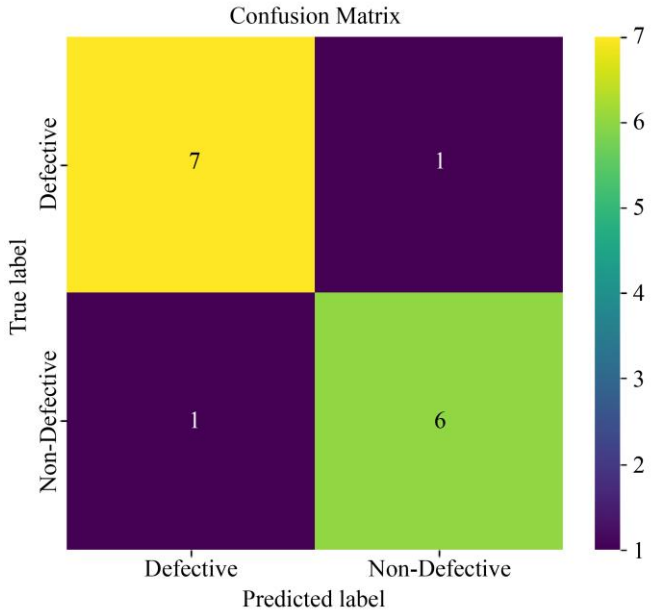

Fig. 8 Confusion matrix for software defect classification

Figure 8 evaluates a classification model's performance for software defect prediction. It shows that the model correctly identified 7 defective items and 6 non-defective items, while it misclassified 1 defective item as non-defective and 1 non-defective item as defective. This matrix provides a clear visualization component.

Table 1. Performance COMPARISON of methods

Method	Accuracy	Precision	Recall	F1 Score
CNN + MLP	85	84	77	83
RNN	78	76	89	78
Rule-Based Algorithm	81	79	73	81
Proposed	92	90	89	90

Table 1 shows the performance of the methods based on Code2Vec and LinkNet-BiLSTM using score-level fusion in the classification of software defects. The proposed method outperforms all metrics because its accuracy is 92%, which shows that it is highly efficient for correctly classifying defects. A precision of 90% means that it has a low false positive rate, and the recall is 89%, which tends to hint at the capacity to find actual defects. The F1 score stands at 90%, and the balance between precision and recall speaks for its robustness in handling complex tasks concerning defect classification. Therefore, This hybrid model incorporates the best features of advanced representation and sequence learning, surpassing the traditional method of CNN + MLP, RNN, and Rule-Based Algorithms, setting a new benchmark in software defect detection.

5.5. Discussion

The proposed hybrid deep learning model has enormous potential for accuracy improvements due to integration between the Code2Vec model, Improved LinkNet, and Bidirectional LSTMs. This principle makes it possible for defect-relevant characteristics to be ascertained on a sound basis, given the capability of the Code2Vec model to translate the raw form of the source code into semantic and syntactic vectors. LinkNet enhances the capture of structural data well, while Bi-LSTM handles sequential data patterns, making it very powerful when handling spatial and temporal data occurrences with the source code. The evaluation process also identifies score-level fusion as one of the critical aspects that could be used to integrate a variety of outputs in a bid to enhance the stability of predictions and reduce variance. The improvement of the validation accuracy following hyper parameter tuning presents a greater emphasis on optimising such hyper parameters as the learning rate and size of batches. However, from some epochs, the validation accuracy does not improve, which strengthens the overfitting concept to address which is used dropout or data augmentation.

6. Conclusion and Future Works

A novel hybrid deep learning model that integrates the Code2Vec model, the Improved LinkNet model, and Bidirectional LSTM with score-level fusion for software defect classification. Addressing traditional defect identification methods' limitations enhances classification, achieving correctness of 92%. The hyperparameter tuning process further improves precision and recall, confirming the approach's effectiveness. However, the model's computational complexity, especially its inability to handle large amounts of data and potential overfitting, highlights areas for further development. The integration of feature extraction and classification elements significantly improves, but the model's generalisation still requires further refinement to be deployed in real-life data implementation. It explores incorporating dropout and cross-validation techniques to mitigate overfitting and enhance generalization. Expanding the dataset with larger, more diverse examples may yield better results, along with using transfer learning for faster training.

Using lightweight architectures and optimization algorithms could also balance accuracy with computational efficiency, making the model more suitable for industrial applications. Finally, the model could be extended to predict hardware or network defects for broader defect prediction capabilities. This work proposes combining Code2Vec, the Improved LinkNet model, and the Bidirectional LSTM with score-level fusion. Eliminating the drawbacks of traditional defect identification helps with classification in the proposed model, where the accuracy rate is 92 %. Subsequently, the hyperparameter tuning process enhances the model accuracy and the recall rate, providing strong evidence for the proposed strategy. However, some problems that still need improvement are identified, such as increasing the model's applicability to extensive data and preventing overfitting. The combination of feature extraction and classification elements enhances the defect prediction, though the grand application of the model still needs enhancements for practical use. The subsequent studies could investigate dropout and cross-validation strategies for obtaining better non-oversensitive generalization. Using larger images can also increase the coverage of a dataset, which, combined with a transfer learning method, can help boost the efficiency of training on the images. Furthermore, the fact that the model has been developed with lightweight architectures and optimization algorithms could support a functional accuracy-complexity trade-off to make it more applicable in industry. The model can provide predictions for hardware or network defects and more generic predictions.

References

[1] Kun Zhu et al., "Software Defect Prediction based on Enhanced Metaheuristic Feature Selection Optimization and a Hybrid Deep Neural Network," *Journal of Systems and Software*, vol. 180, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[2] Ning Li, Martin Shepperd, and Yuchen Guo, "A Systematic Review of Unsupervised Learning Techniques for Software Defect Prediction," *Information and Software Technology*, vol. 122, pp. 1-18, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[3] Wei Zheng et al., "Interpretability Application of the Just-in-Time Software Defect Prediction Model," *Journal of Systems and Software*, vol. 188, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang, "Software Defect Prediction based on Gated Hierarchical LSTMs," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 711-727, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Fanqi Meng, Wenying Cheng, and Jingdong Wang, "Semi-Supervised Software Defect Prediction Model based on Tri-Training," *KSII Transactions on Internet and Information Systems*, vol. 15, no. 11, pp. 4028-4042, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Elena N. Akimova et al., "A Survey on Software Defect Prediction using Deep Learning," *Mathematics*, vol. 9, no. 11, pp. 1-14, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[7] Geanderson Esteves et al., "Understanding Machine Learning Software Defect Predictions," *Automated Software Engineering*, vol. 27, pp. 369-392, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[8] Amirabbas Majd et al., "SLDeep: Statement-level Software Defect Prediction using Deep-Learning Model on Static Code Features," *Expert Systems with Applications*, vol. 147, pp. 1-14, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[9] Fei Deng et al., "Accelerating Magnetotelluric Forward Modeling with Deep Learning: Conv-BiLSTM and D-LinkNet," *Geophysics*, vol. 88, no. 2, pp. E69-E77, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Emin Borandag, "Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques," *Applied Sciences*, vol. 13, no. 3, pp. 1-21, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[11] Iqra Batool, and Tamim Ahmed Khan, "Software Fault Prediction using Deep Learning Techniques," *Software Quality Journal*, vol. 31, pp. 1241-1280, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[12] Wahaj Alkaberi, and Fatmah Assiri, "Predicting the Number of Software Faults using Deep Learning," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13222-13231, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] H. Das et al., "Enhancing Software Fault Prediction Through Feature Selection With Spider Wasp Optimization Algorithm," *IEEE Access*, vol. 12, pp. 105309-105325, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[14] Sonika Chandrakant Rathi et al., "Empirical Evaluation of the Performance of Data Sampling and Feature Selection Techniques for Software Fault Prediction," *Expert Systems with Applications*, vol. 223, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[15] Fengyu Yang et al., "Interpretable Software Defect Prediction Incorporating Multiple Rules," *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Taipa, Macao, pp. 940-947, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Ke Shi et al., "PathPair2Vec: An AST Path Pair-Based Code Representation Method for Defect Prediction," *Journal of Computer Languages*, vol. 59, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[17] Abhishek Chaurasia, and Eugenio Culurciello, "Linknet: Exploiting Encoder Representations for Efficient Semantic Segmentation," *2017 IEEE Visual Communications and Image Processing (VCIP)*, St. Petersburg, FL, USA, pp. 1-4, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[18] Ahmed Bahaa Farid et al., "Software Defect Prediction using Hybrid Model (CBIL) of Convolutional Neural Network (CNN) and Bidirectional Long Short-Term Memory (Bi-LSTM)," *PeerJ Computer Science*, vol. 7, pp. 1-22, 2021. [CrossRef] [Google Scholar] [Publisher Link]