Revolutionizing Software Project Development: A CNN-LSTM Hybrid Model for Effective Defect Prediction

Selvin Jose G[1], Dr. J Charles[2]
Department of Computer Science, Noorul Islam Centre for Higher Education, Kumaracoil, Tamil Nadu, India[1]
Department of Software Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, Tamil Nadu, India[2]

***Abstract*—Within the domain of software development, the practice of software defect prediction (SDP) holds a central and critical position, significantly contributing to the efficiency and ultimate success of projects. It embodies a proactive approach that harnesses data-driven techniques and analytics to preemptively identify potential defects or vulnerabilities within software systems, thereby enhancing overall quality and reliability while significantly impacting project timelines and resource allocation. The efficiency of software development projects hinges on their ability to adhere to deadlines, budget constraints, and deliver high-quality products. SDP contributes to these objectives through various means. This paper introduces a novel SDP model that harnesses the combined capabilities of Convolutional Neural Networks (CNNs) and Long Short Term Memory (LSTMs) unit. CNNs excel at extracting features from structured data, enabling them to discern patterns and dependencies within code repositories and change histories. LSTMs, conversely, excel in handling sequential data, which is pivotal for capturing the temporal aspects of software development and tracking the evolution of defects over time. The outcomes of the proposed CNN-LSTM hybrid model showcase its superior predictive performance. Simulation results affirm the substantial potential of this model to bolster the efficiency and reliability of software development processes. As technology advances and data-driven methodologies become increasingly prevalent in the software industry, the integration of such hybrid models presents a promising avenue for continually elevating software quality and ensuring the triumph of software projects. In summary, the utilization of this innovative SDP model offers a transformative approach to efficient software development, positioning it as a vital tool for project success and quality assurance.**

Keywords*—Data driven software development; proactive defect identification; software quality; predictive analytics; software defect prediction; artificial intelligence; long short term memory

I. Introduction

In the modern world, software plays an essential role in every aspect of our daily existence. It includes defense, automobile, healthcare, insurance, finance, banking, telecommunication, government administration sectors. In other words, the normal functionality of these sectors gets affected with the software failure. Technical and managerial issues are the two different issues normally emerge during the software development process. Thirty percent of project failures occur mainly due to technical issues and 70% are management issues [1]. Some of the problems related to managerial issues are insufficient risk management, customer buy-in, limited project resources and inaccurate project structure etc. However, low product delay, high expense, and schedule delay are the issues encountered during software program development. Prior to the analysis of software project risk, an efficient risk mitigation scheme should be developed by the program developer. Based on the accurate management of risk, the success of the project can be determined [2].

SDP is a crucial aspect of modern software development, aimed at improving the effectiveness and efficiency of software projects. In an era where software plays an increasingly pivotal role in our daily lives, organizations strive to deliver high-quality software products while minimizing the time and resources invested in debugging and maintenance. In this context, SDP emerges as an indispensable tool for the modern software development landscape, fostering both agility and the delivery of robust software products. Any fault, error, mistake, in a computer program, or a defect or bug in the software can cause unexpected or inaccurate results which are otherwise called a software defect. In order to enhance software quality, high-risk components must be detected as soon as possible [3].

Software defects can lead to an increase in both the cost and time required for delivering the expected end product. Also identification and rectification of defects is a highly waste of time and a costly software process [4]. One of the persistent challenges within the Software Development Lifecycle (SDLC) has been the ability to predict and identify defects during the initial phases of a project. In the current situation, development of a fault-free software which is highly reliable is a difficult task, as the problems for which software is developed is more complex and the domains that are involved are constantly increasing to constraints such as uncertainty and development processes that are complex [5].

At first, a collection of project data takes place from the software repositories. From the data, factors are calculated. The locations are predicted through models, which have a better potential for the defects contained. Ultimately, using prediction models, various measures are evaluated, such as precision, recall, and explanative power [6].

SDP using Deep Learning (DL) holds immense significance in the realm of software project development. In an era where software systems underpin virtually every aspect of modern life, ensuring their reliability and robustness is paramount. Deep learning methodology, leveraging their capacity to analyzes extensive datasets and detect subtle patterns, present a compelling approach for preemptively recognizing and addressing potential flaws before they escalate into severe problems. By harnessing the power of DL, software development teams can enhance their efficiency, reduce maintenance costs, and deliver higher-quality products to users, ultimately contributing to the successful and sustainable advancement of software projects in an increasingly interconnected world. The major contribution of the proposed work includes:

- Software defect prediction utilizing CNN-LSTM hybrid model.
- To employ CNN in feature extraction from structured data, particularly in code repositories and change histories.
- To employ LSTM in handling sequential data, emphasizing their pivotal role in capturing the temporal aspects of software development and facilitating the tracking of defect evolution over time.

The paper is systematized as follows: Section II offers are view of the existing literature and identifies areas where further research is needed. Section III outlines the methodology in detail. In Section IV, the comprehensive results of the suggested approach are discussed. Finally, in Section V and Section VI, the paper concludes with a discussion and summary respectively.

II. Background

A. Literature Review

Lei Qiao et al. [7] introduced an innovative methodology employing DL approaches for the anticipation of software system defects. This novel approach involves training a DL approach to predict the number of defects in software. Notably, when compared to widely adopted approaches such as Support Vector Regression, Feature-based Support Vector Regression, and Decision Tree Regression, the suggested method demonstrated a substantial enhancement in performance on established datasets. The improvement is notably reflected in a notable reduction in mean square error, ranging from 3% to 13%, and an augmentation in the squared correlation coefficient.

Pan et al. [8] introduced a range of CodeBERT models, specifically designed for SDP. The proposed research involved conducting empirical studies to assess the effectiveness of these approaches in cross-version and cross-project SDP scenarios. The findings demonstrated that leveraging pre-trained CodeBERT models led to enhanced prediction accuracy and time savings. Additionally, incorporating sentence-based and keyword-based prediction approaches further improved the effectiveness of pre-trained neural language frameworks in the context of SDP.

Geanderson Esteves et al. [9], delved into the realm of SDP models, harnessing the power of an efficiently implemented XGBoost variant, known as US-XGBoost. This endeavor generated a multitude of random models, each meticulously assessed for the accuracy and interpretability. The key take away from the findings is that SDP is inherently project-specific. This means that the features constituting the most effective models can significantly differ from one project to another. Hence, comprehending the determinants behind model decisions becomes particularly vital.

Lakshmi Prabha and N. Shivakumar [10] introduced a novel hybrid model that addresses the challenge of classifying massive datasets accurately. The proposed approach combines feature reduction using Principal Component Analysis (PCA) with an overall probability application to minimize data loss during PCA processing. The approach further employed a neural network classification method for program bug detection. The simulation results demonstrated the model's impressive efficiency, achieving an outstanding 98.70 percent Area under the Curve (AUC) accuracy, marking a substantial advancement over existing models.

Hao Wang et al. [11] introduced GH-LSTMs, a novel DL framework for detecting potential code defects within software modules. GH-LSTMs leverage hierarchical LSTM architecture to simultaneously extract semantic and traditional features. A gated merge mechanism was employed to dynamically optimize the fusion of these features. Subsequently, a fully connected layer utilizes the combined features for within-project defect prediction. Remarkably, GH-LSTMs outperform existing methods in terms of F-measure, particularly in non-effort-aware cases.

Bilal Khan et al. [12] presented a comprehensive analysis of seven widely employed Machine Learning (ML) approaches applied to SDP. These approaches encompass SVM, J48, RF, MLP, RBF, HMM, and CDT. The evaluation of these methods utilized various performance metrics, including MAE, RAE, RMSE, RRSE, recall, and accuracy. The findings from the experiments revealed that NB and SVM exhibited superior performance in terms of minimizing MAE and RAE, respectively.

Shuo Feng et al. [13], delved into the robustness of SMOTE-based oversampling methods. This work not only probed the stability of these techniques but also introduced a set of novel and stable SMOTE-based oversampling strategies aimed at enhancing the reliability. These stable techniques minimize the inherent randomness in SMOTE by sequentially selecting defective instances, utilizing a distance-based approach for choosing neighbor instances, and ensuring an evenly distributed interpolation process. The proposed approach supported the findings with both mathematical proofs and empirical investigations across 26 datasets using four common classifiers. The simulation results demonstrated that the effectiveness of stable SMOTE-based oversampling approaches surpasses that of traditional SMOTE-based approaches in terms of stability and effectiveness.

Somya Goyal [14] introduced a pioneering Neighborhood-based Under-Sampling (N-US) algorithm to address the challenge of class imbalance. The study aims to showcase the efficacy of this N-US framework in enhancing accuracy for predicting defective modules. The experimental results revealed that the N-US approach successfully reduces the dataset size by 17.29% and lowers the Imbalance Ratio (IR) by 19.73%. Consequently, it plays a vital role in augmenting classifier performance.

Cong Jin [15] introduced an innovative distance metric learning framework that leverages cost-sensitive learning (CSL) to mitigate the challenges posed by class-imbalanced datasets. This novel method, initially developed to address class imbalance, assigns distinct weights to individual training classes. Subsequently, this CSL-based distance metric learning is integrated into the large margin distribution machine (LDM) to take over the conventional kernel function. Empirical results indicated that these enhancements enable CS-ILDM to exhibit not only excellent predictive performance but also the lowest misprediction cost.

Kun Zhu et al. [16] introduced an innovative feature selection algorithm called EMWS, which optimally chooses a compact set of closely related features tailored to each software project. This approach effectively harnesses the local search capabilities of simulated annealing to augment the relatively weaker exploitation performance of the Whale Optimization Algorithm (WOA) while simultaneously capitalizing on WOA's strong global search abilities to enhance SA's exploration capabilities. A hybrid deep neural network model was also proposed. Empirical results substantiate that EMWS and WSHCKE consistently outperform various methods in various experiments.

Ruba Abu Khurma et al. [17] introduced the Island Model as an enhancement to the Binary Moth Flame Optimization (BMFO) algorithm for addressing the Feature Selection problem in the context of SDP. This innovative approach segments the moth population into multiple islands, facilitating feature sharing among them through migration. This technique serves to bolster solution diversity and govern algorithm convergence. The experiments involved assessing the performance of KNN, NB and SVM classifiers with and without FS, using BMFO-FS, and employing Is BMFO-FS. Notably, across three experiments, the SVM classifier consistently outperformed others, closely followed by the NB classifier.

Shuo Feng et al. [18] introduced a novel oversampling approach known as Complexity-based Oversampling Technique (COSTE). Instead of relying on inter-instance distances, COSTE assesses instance complexity to guide the selection of candidates for generating synthetic instances. The study evaluated COSTE's effectiveness against four other oversampling techniques using various classifiers, including, KNN, MLP, SVM and RF, across 23 imbalanced datasets. Remarkably, the simulation findings consistently demonstrated that COSTE outperformed the other methods across all performance metrics, highlighting its superior performance.

Shiqi Tang et al. [19] introduced TSboostDF, an innovative transfer-learning algorithm designed to address the complex problem of CPDP (Cross-Platform Domain Prediction). TSboostDF effectively combines the BLS sampling method, which considers the sample's weight, with transfer-learning techniques to mitigate the limitations commonly associated with conventional CPDP algorithms. This novel approach has been demonstrated to outperform other CPDP algorithms that rely on transfer-learning methods, highlighting its superior performance in resolving this challenging problem.

Liu Yang et al. [20] introduced an innovative hybrid algorithm that combines the strengths of SSA and PSO. This research involved a comprehensive analysis of the merits and limitations of swarm intelligence algorithms, aiming to devise strategies for enhancement. Notably, the empirical findings demonstrated that the hybrid approach integrating SSA and PSO, as presented in this work, significantly enhances the precision of software reliability model estimation and forecasting. Specifically, the proposed study focused on estimating and predicting software defects using the well-known G-O model. Furthermore, a fitness function was introduced, which is capable of effectively managing the parameter 'b' during initialization by leveraging the maximum likelihood formula.

An algorithm was presented by Nassif et al. [21] that aims to accomplish two significant tasks: learning and prediction. This approach has a high efficiency for other issues, such as software defect prediction, while being widely utilized in information retrieval. In this paper, two common output metrics namely bug density bug count were used as goal variables to compare various models. Additionally, it looked at how eight models with Grid Search optimization were affected by the use of imbalance learning and feature selection. The FPA scores of the bug density results have significantly improved with the usage of imbalance learning; however, the improvement in the bug count results has not been as great. Last but not least, applying feature selection with LTR decreased the bug density metric's FPA score but had no effect on the bug count findings.

B. Research Gap

SDP models offer valuable insights and benefits, but they also come with several limitations. Some of the major limitations of existing SDP models are discussed below. These models heavily rely on historical data, which may be incomplete, inconsistent, or biased. Poor data quality can induce to inaccurate predictions. Software defect datasets often have imbalanced class distributions, with a small number of defective occurrences compared to non-defective ones. This can lead to model bias and lower predictive accuracy. Software systems, tools, and development practices evolve over time. Models trained on historical data may not effectively adapt to new technologies and practices. Creating relevant features from code repositories and other software data is a complex and manual process. Feature engineering can be time-consuming and error-prone. Complex ML models can over fit the training data, making them less generalizable to new projects or software environments. These models identify correlations but not necessarily causation. Identifying the root causes of defects often requires domain expertise and additional analysis. Models may not be transferable to different software domains or projects due to the unique characteristics of each project. Software is continuously evolving, and defects can emerge or be resolved after the training data was collected, making predictions less accurate. Bias in training data and predictions can lead to discrimination or unfair treatment in software development processes. Training and deploying sophisticated ML models can necessitate remarkable computational resources and expertise, which may not be readily available to all development teams. Addressing these limitations requires careful consideration and often a combination of techniques, including data preprocessing, model selection, and ongoing monitoring and validation of the model's performance. Despite these challenges, SDP models have the potential to significantly improve software development processes when appropriately applied and maintained.

III. Materials and Methods

A CNN- LSTM based hybrid DL model is developed and analyzed for SDP for effective software project development. The detailed block schematic of the suggested work is illustrated in Fig. 1. The initial step of the work involves the dataset collection. It is followed by data preprocessing techniques. The preprocessed data is separate into training set and test set. The proposed hybrid model is trained and validated utilizing the training data and test data. Finally, the performance of the SDP model is analyzed.

A. *Dataset Description*

The proposed system utilizes SDP dataset collected from Open ML, an online platform and repository for ML datasets. The dataset contains 31 features of 224 instances. In this paper, the binary classifier is developed to predict software defects based on 31 inputs.

B. *Units Data Preprocesing and Exploratory Data Analysis*

Data preprocessing is a pivotal stage in data analysis, encompassing the tasks of cleansing, restructuring, and organizing raw data to make it appropriate for analysis or ML model training. The quality and efficacy of a learning model are substantially influenced by proper data preprocessing. Key techniques involved in this process include data cleaning, data transformation, handling missing values, addressing duplicate entries, and managing outliers. Among these techniques, the management of missing values stands out as a critical step. It involves handling data points that lack complete or relevant information. Various strategies can be employed for this purpose. One approach is to eliminate rows or columns with an excessive number of missing values, particularly if they do not significantly contribute to the analysis. An alternative method is imputation, which involves filling in the gaps with estimated or calculated values based on the data's distribution. For numerical data, this can involve mean, median, or mode imputation, ensuring that the dataset is more robust and suitable for analysis or modeling.

Exploratory Data Analysis (EDA) serves as a vital initial step in the data analysis process, where data analysts and scientists employ both visual and statistical methods to delve into a dataset. Its primary goal is to unveil patterns, relationships, anomalies, and insights within the data. EDA entails a range of techniques, including data visualization tools such as histograms, scatter plots, and box plots, in combination with summary statistics like mean, median, standard deviation, and more. This multifaceted approach allows for a comprehensive understanding of data distribution, the detection of outliers, an evaluation of data quality, and the development of an intuitive grasp of the dataset's underlying structure. In practice, EDA plays a pivotal role in hypothesis formulation, guiding subsequent analytical processes, and informing decisions related to data preprocessing and modeling strategies. Ultimately, it aids in the discovery of valuable information and concealed patterns within the data. Summary statistics, which provide a concise summary of key dataset characteristics, include measures such as mean, median, mode, standard deviation, variance minimum and maximum values, quartiles including the first, second or median, and third quartiles, and counts or proportions for categorical variables. The summary statistics of SDP dataset is illustrated in Fig. 2.

EDA relies heavily on the strategic use of graphical representations to delve into a dataset. Through the construction of diverse charts, plots, and graphs, including histograms, scatter plots, box plots and heatmaps, it becomes possible to visually examine data distributions, reveal underlying patterns, identify anomalies, and grasp the interplay between variables. The data visualization of counts of classes in the dataset is visualized in Fig. 3.

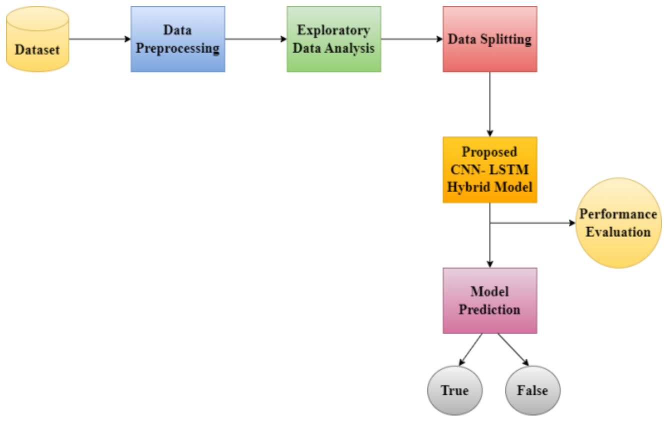
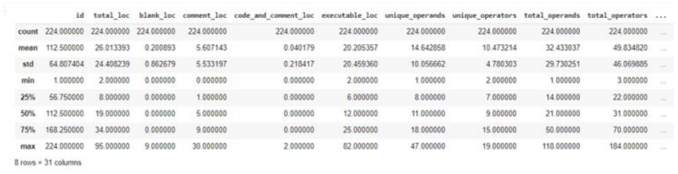


Fig. 1. Block Schematics of Proposed SDP model.



	id	total_loc	blank_loc	comment_loc	code_and_comment_loc	executable_loc	unique_operands	unique_operators	total_operands	total_operators	...
count	224.000000	224.000000	224.000000	224.000000	224.000000	224.000000	224.000000	224.000000	224.000000	224.000000	...
mean	112.500000	26.013393	0.200893	5.607143	0.040179	20.205357	14.642858	10.473214	32.433037	49.834820	...
std	64.807404	24.408239	0.862679	5.533197	0.218417	20.459360	10.056662	4.780303	29.730251	46.069885	...
min	1.000000	2.000000	0.000000	0.000000	0.000000	2.000000	1.000000	2.000000	1.000000	3.000000	...
25%	56.750000	8.000000	0.000000	1.000000	0.000000	6.000000	8.000000	7.000000	14.000000	22.000000	...
50%	112.500000	19.000000	0.000000	5.000000	0.000000	12.000000	11.000000	9.000000	21.000000	31.000000	...
75%	168.250000	34.000000	0.000000	9.000000	0.000000	25.000000	18.000000	15.000000	50.000000	70.000000	...
max	224.000000	95.000000	9.000000	30.000000	2.000000	82.000000	47.000000	19.000000	118.000000	184.000000	...

8 rows × 31 columns

Fig. 2. Summary statics of SDP dataset.

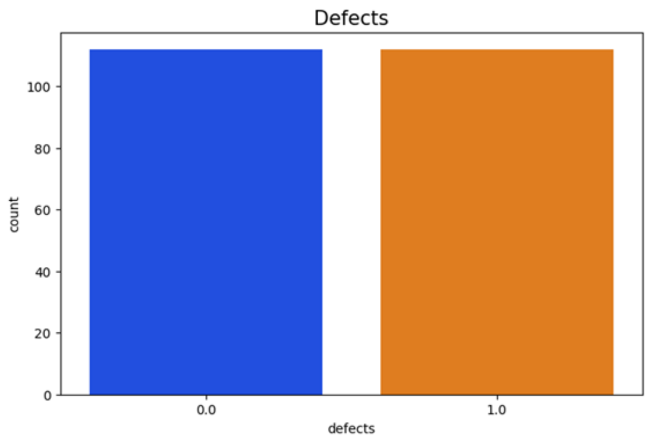


Fig. 3. Data visualization of SDP dataset.

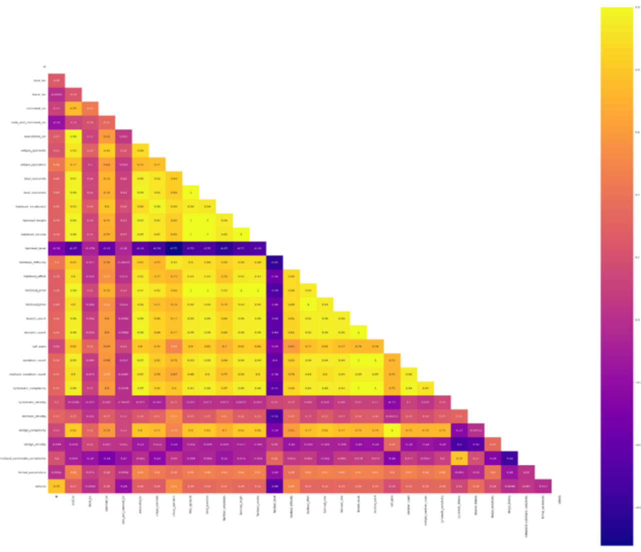


Fig. 4. Heatmap visualization of dataset.

Heatmap visualization serves as a powerful graphical tool for depicting data, representing information within a matrix by employing a range of colors. Its primary utility lies in the effective exploration of complex datasets, as each individual cell in the matrix corresponds to a specific data point, and the color intensity within these cells conveys the underlying data values, often using a color spectrum to illustrate the transition from lower to higher values. Fig. 4 illustrates the heatmap visualization of dataset.

C. SDP Model using CNN-LSTM Hybrid Model

The proposed defect prediction model is a hybrid system that leverages both CNNs and LSTM neural networks. This innovative approach aims to enhance the accuracy of identifying defects within software. CNNs are employed to detect spatial patterns in the code, such as relationships between different code segments, while LSTMs excel at modeling sequential dependencies over time. By amalgamating these two architectural components, the model becomes capable of understanding both localized and global patterns in the code base, enabling it to effectively pinpoint software defects. The approach involves encoding code as input sequences, which are then processed by CNN layers to capture spatial characteristics and subsequently by LSTM layers to capture temporal relationships. The resulting hybrid model offers a more precise and comprehensive defect prediction, which, in turn, supports the development of more dependable software systems.

A CNN is a type of deep ANN that emulates the human visual perception process, making it particularly effective for analyzing visual data [22]. CNNs leverage various multi-layer perceptron algorithms to reduce the need for extensive preprocessing of input data, making them widely adopted in the field of DL. CNNs represent one of the most prevalent neural network architectures, typically comprising millions of interconnected neurons organized into hierarchical layers, as illustrated in Fig. 5.

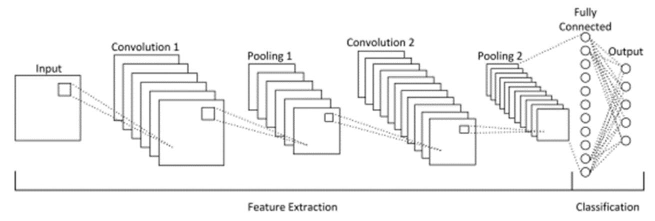


Fig. 5. Basic architecture of CNN.

CNN consist of three fundamental layers: convolutional, pooling, and fully-connected layers. The hidden layers within the convolutional and FC layers play a pivotal role in accessing the network's capacity to learn. The depth of a CNN is defined by the number of layers it comprises, and the deeper these layers are, the more intricate and abstract features they can extract from the input data, especially in the context of high-resolution images [23]. Within the CNN processing pipeline, the neurons in the input layer respond to visual stimuli, initiating the feature extraction process. The major aim of the convolutional layer is to capture these image features and propagate them to the subsequent hidden layers for computation, culminating in the extraction of results from the output layer. Activation functions often act as intermediaries between hidden layers, facilitating the transfer of valuable and essential information to inform the subsequent layers in the network.

The Long Short-Term Memory unit a prominent component of DL belongs to the family of Recurrent Neural Networks (RNNs) [24]. This specialized RNN excels in understanding and capturing intricate order dependencies within sequence prediction tasks. It is specifically designed to manage long-term relationships and tackle challenging problems, particularly those where the input order plays a pivotal role. Over time, numerous variations of Deep RNNs have been devised to combat issues related to vanishing and exploding gradients. Among these, the LSTM network stands out as a unique solution. It achieves its exceptional capabilities by employing distinct activation functions for each of its gates, allowing it to remember essential information from the past while efficiently discarding irrelevant data. Furthermore, LSTM incorporates an internal cell state vector, which serves as a practical representation of the network's retained knowledge from prior inputs. The LSTM unit employs three distinct gates: Forget Gate (f), Input Gate (i) and the Output Gate (o). Fig. 6 illustrates the core structural elements of an LSTM unit.

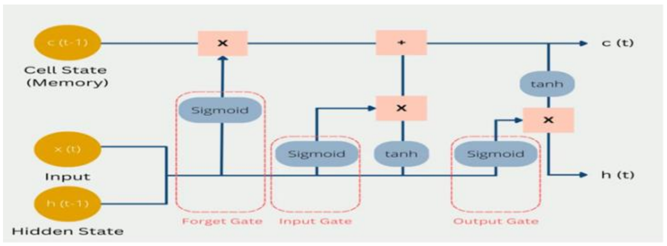


Fig. 6. Basic architecture of LSTM.

In a hybrid CNN-LSTM model, the initial CNN stage operates on the input sequence, effectively pinpointing spatial patterns or local features at each point in time. The resulting CNN layer outputs are subsequently fed into the LSTM component, which is responsible for modeling the temporal dependencies and capturing the sequential patterns within the data. This combined approach enables the model to effectively assimilate both local and global information, making it a robust choice for tasks that entail complex spatial and temporal interactions in sequential data. The architectural representation of the proposed model is visually depicted in Fig. 7.

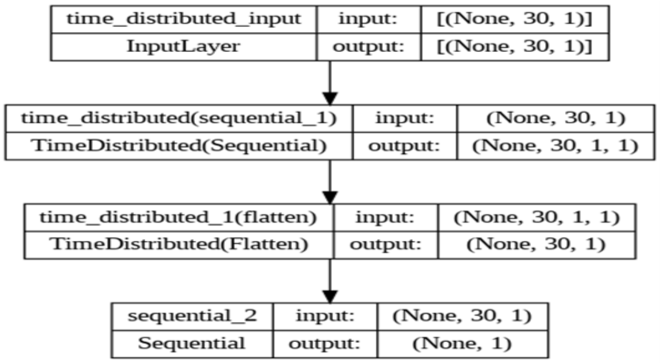

time_distributed_input | input: | [(None, 30, 1)]
InputLayer | output: | [(None, 30, 1)]
time_distributed(sequential_1) | input: | (None, 30, 1)
TimeDistributed(Sequential) | output: | (None, 30, 1, 1)
time_distributed_1(flatten) | input: | (None, 30, 1, 1)
TimeDistributed(Flatten) | output: | (None, 30, 1)
sequential_2 | input: | (None, 30, 1)
Sequential | output: | (None, 1)

Fig. 7. Proposed model architecture.

The suggested hybrid model begins with a dense layer featuring 100 neurons and a Relu activation function. This is succeeded by two additional dense layers, one with 50 neurons and the other with 25 neurons, both utilizing ReLU activation. The final dense layer has of a single neuron with a sigmoid activation function. Subsequently, an LSTM layer with 64 units is incorporated, followed by the application of a dropout layer with a 0.5 dropout rate to avoid over fitting. This is succeeded by another LSTM layer with 32 units. Finally, the model concludes with a dense layer consists a single neuron and a sigmoid activation function. This hybrid architecture combines a CNN with a recurrent neural network (RNN) using Time Distributed layers. The Time Distributed layers enable the application of feed forward network and flatten operations across each time step of the input sequence, effectively leveraging both spatial and temporal information for sequence-based tasks.

IV. Experimental Analysis

A. Hardware and Software Setup

The proposed system utilizes SDP dataset contains 31 attributes. Google Collaboratory and Microsoft windows 10 are chosen in this research to ensure a stable computing experience. The system is equipped with an Intel Core i7-6850K 3.60 GHz 12- core processor, one NVIDIA Geforce GTX 1080 Ti GPU 2760 4MB. The dataset is split into two sets: training set (70%) and test set (30%). The entire procedure made use of Python and TensorFlow. The 'Adam' optimization function was used in the proposed model. The binary crossentropy is used as the loss function. The training process involved using a batch size of 32 for a total of 750 epochs. Finally, the proposed model predicts software defects as TRUE or FALSE.

B. Result

The effectiveness of the suggested SDP model underwent an assessment using several key performance parameters, including accuracy, precision, recall, F1-score, specificity, and ROC AUC. Accuracy, a statistical measure, was employed to gauge the model's classification performance, representing the percentage of accurately predicted instances out of the entire dataset.

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \quad (1)$$

Precision is a parameter utilized to find the model's capability to accurately predict positive outcomes. It quantifies the ratio of true positive predictions to all positive predictions, encompassing both accurate positive predictions and false positives.

$$Precision = \frac{(TP)}{(TP+FP)} \quad (2)$$

Recall is a parameter that assesses a model's capacity to efficiently detect all pertinent examples of a specific class in a dataset. It quantifies the ratio of true positive predictions to the total number of actual occurrences belonging to that class.

$$Recall = \frac{(TP)}{(TP+FN)} \quad (3)$$

The F1-Score serves as a metric employed to evaluate how well precision and recall are balanced in binary classification tasks.

$$F1 - Score = 2 * \frac{(Precision * Recall)}{(Precision + Recall)} \quad (4)$$

Specificity refers to the degree of precision or accuracy in targeting a particular characteristic, attribute, or aspect within a given context. The classification report of the proposed SDP model after simulation is tabulated in Table I.

An accuracy plot is a graphical representation that shows how well a model's predictions match the actual outcomes or labels in a dataset. It is a visual tool utilized to assess the effectiveness of a model, with the x-axis usually representing different iterations or epochs of training, and the y-axis indicating the accuracy of the model's predictions. The accuracy plot of the suggested prediction model is visualized in Fig. 8.

TABLE I. PERFORMANCE EVALUATION OF PROPOSED SDP MODEL

Performance Metrics	Obtained Results
Accuracy	95.58 %
Precision	96.66 %
Recall	93.54 %
F1- Score	95.08 %
ROC AUC	0.9542
Specificity	97.29 %
Cohens Kappa	0.9108

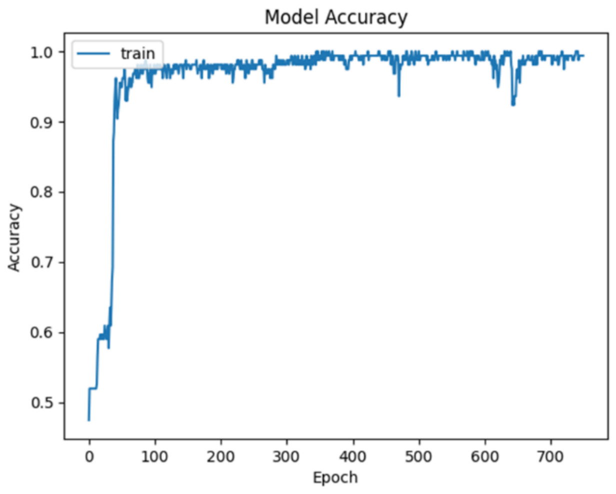


Fig. 8. Accuracy plot of proposed SDP model.

Loss plot typically shows how the loss function, a measure of the error between the predicted and actual values, changes over time as the model learns from the training data. The loss plot is a critical tool for assessing the training process. The loss plot of the proposed SDP model is visualized in Fig. 9.

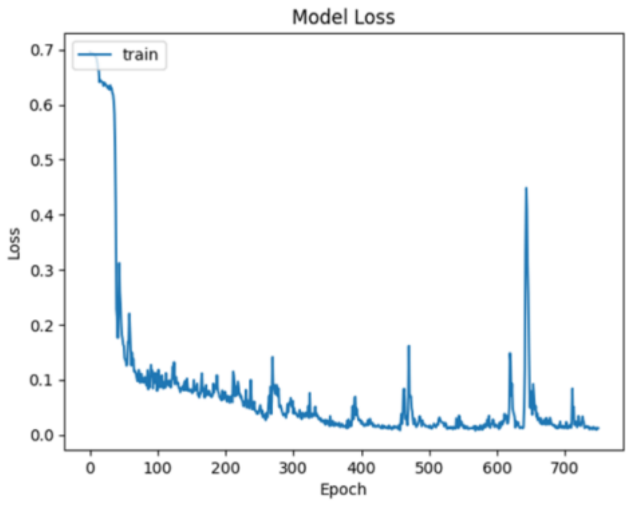


Fig. 9. Loss plot of proposed SDP model.

A confusion matrix is a visual representation of a classification model's performance that helps assess its accuracy and error rates. This grid is structured in such a way that it assigns actual class labels to its rows and predicted class labels to its columns. The cells along the diagonal of the matrix account for the accurate predictions, encompassing both true positives and true negatives. Meanwhile, any discrepancies outside the diagonal signify errors, including false positives and false negatives. The confusion matrix obtained for the proposed SDP model is illustrated in Fig. 10.

The Receiver Operating Characteristic (ROC) curve is a visual tool utilized to assess the effectiveness of models. It provides a graphical representation of how well a model can differentiate between positive and negative classes by depicting the balance between its true positive rate and false positive rate across various threshold settings. When examining a ROC curve, a flawless model would closely follow the upper-left corner of the plot, demonstrating high sensitivity and minimal false positives, whereas random guessing would produce a diagonal line running from the lower-left to the upper-right. The Area Under the ROC Curve (AUC-ROC) serves as a succinct metric summarizing the model's overall performance, with greater values indicating superior discriminatory capabilities. The ROC curve is visualized in Fig. 11. Table II shows the performance comparison of proposed model with existing model.

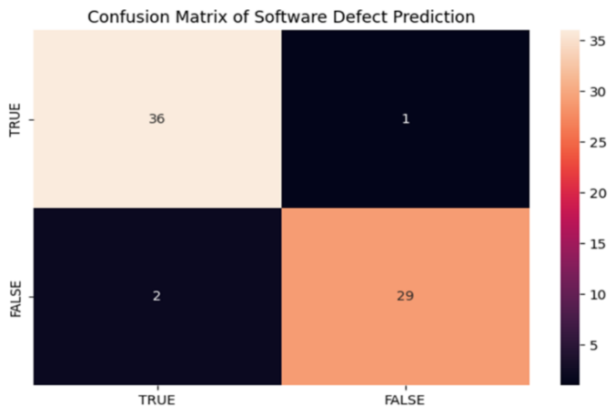


Fig. 10. Confusion matrix of proposed SDP model.

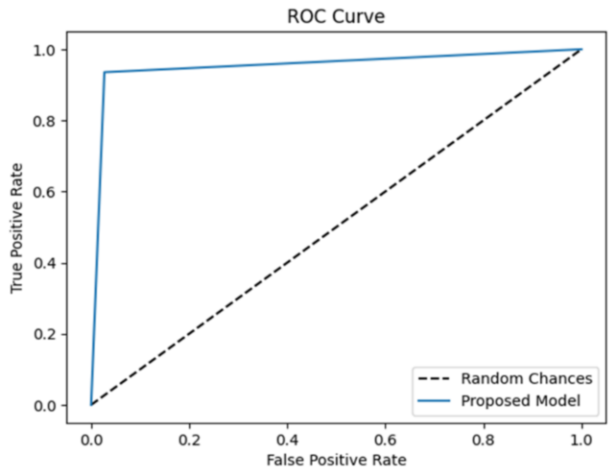


Fig. 11. ROC curve.

TABLE II. PERFORMANCE COMPARISON OF PROPOSED MODEL WITH EXISTING MODELS

Author &Reference	Methodology	Result
12	RF	88.32%
14	KNN	94.6
16	CNN and kernel extreme learning machine	93.5
17	moth flame optimization	80%
20	Hybrid Particle Swarm Optimization and Sparrow Search Algorithm	88%
Proposed model	**CNN+LSTM**	**95.58%**

V. Discussion

In the realm of software defect prediction, the findings of this study contribute valuable insights into the identification and mitigation of software defects. Through a comprehensive analysis and employing advanced learning algorithms, the study successfully establishes robust models for predicting potential defects in software development. The results reveal key predictors and patterns associated with defect occurrence, shedding light on critical areas that warrant attention during the software development life cycle. Feature extraction from structured data making them adept at identifying patterns and dependencies within code repositories, change histories. On the other hand, are well-suited for handling sequential data, which is crucial in capturing temporal aspects of software development and tracking the evolution of defects over time.

Moreover, the study's exploration of different feature sets and model evaluation techniques enhances the reliability of the proposed defect prediction models. The simulation results demonstrated that the adoption of a CNN-LSTM hybrid model for SDP has the potential to significantly contribute to more efficient and reliable software development processes.

VI. Conclusion

SDP plays a pivotal role in ensuring efficient development in software projects. This approach is proactive in nature, utilizing data-driven methods and analytics to detect possible flaws or weaknesses in software systems prior to the escalation into critical problems. This contributes not only to the improvement of the software's overall quality and dependability but also exerts a substantial influence on project schedules and resource allocation. Efficient software development projects are characterized by their ability to meet deadlines, stay within budget constraints, and deliver a high-quality product. SDP contributes to these goals in several key ways. This paper presented a SDP model utilizing both the benefits of CNN and LSTM. This approach leverages the power of CNN and LSTM to address the challenges associated with identifying and mitigating software defects, ultimately contributing to the improvement of software quality and project timelines. The CNN-LSTM hybrid model combines the strengths of both convolutional and recurrent neural networks. CNNs excel in feature extraction from structured data, making them adept at identifying patterns and dependencies within code repositories and change histories. LSTMs, on the other hand, are well-suited for handling sequential data, which is crucial in capturing temporal aspects of software development and tracking the evolution of defects over time. The proposed prediction model achieved better prediction results. The simulation results demonstrated that the adoption of a CNN-LSTM hybrid model for SDP has the potential to significantly contribute to more efficient and reliable software development processes. As technology continues to advance and data-driven approaches become increasingly prevalent in the software industry, the integration of such models holds promise for continually enhancing software quality and the success of software projects.

References

[1] Verner, J., Sampson, J., & Cerpa, N. (2008, June). What factors lead to software project failure?. In 2008 second international conference on research challenges in information science (pp. 71-80). IEEE.

[2] Sangaiah, AK, Samuel, OW, Li, X, Abdel-Basset, M & Wang, H 2018, 'Towards an efficient risk assessment in software projects-Fuzzy reinforcement paradigm', Computers & Electrical Engineering, vol. 71, pp. 833-846. https://doi.org/10.1016/j.compeleceng.2017.07.022.

[3] Shukla, HS & Verma, DK 2015, 'A Review on Software DefectPrediction', International Journal of Advanced Research in ComputerEngineering & Technology (IJARCET), vol. 4 no. 12, pp. 4387-4394.

[4] Gupta, V, Ganeshan, N & Singhal, TK 2015, 'Developing SoftwareBug Prediction Models Using Various Software Metrics as the BugIndicators', International Journal of Advanced Computer Science &Applications, vol. 1, no. 6, pp. 60-65.

[5] Vashisht, V, Lal, M, Sureshchandar, GS & Kamya, S 2015, 'Aframework for software defect prediction using neural networks',Journal of Software Engineering and Applications, vol. 8, no. 8,pp. 384-394.

[6] Shihab, E 2012, An exploration of challenges limiting pragmaticsoftware defect prediction (Doctoral dissertation, Queen's University).Link:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.467.1447&rep=rep1&type=pdf.

[7] Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning-based software defect prediction. Neurocomputing, 385, 100-110.

[8] Pan, C., Lu, M., & Xu, B. (2021). An empirical study on software defect prediction using codebert model. Applied Sciences, 11(11), 4793.

[9] Esteves, G., Figueiredo, E., Veloso, A., Viggiato, M., & Ziviani, N. (2020). Understanding machine learning software defect predictions. Automated Software Engineering, 27(3-4), 369-392.

[10] Prabha, C. L., & Shivakumar, N. (2020, June). Software defect prediction using machine learning techniques. In 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184) (pp. 728-733). IEEE.

[11] Wang, H., Zhuang, W., & Zhang, X. (2021). Software defect prediction based on gated hierarchical LSTMs. IEEE Transactions on Reliability, 70(2), 711-727.

[12] Khan, B., Naseem, R., Shah, M. A., Wakil, K., Khan, A., Uddin, M. I., & Mahmoud, M. (2021). Software defect prediction for healthcare big data: an empirical evaluation of machine learning techniques. Journal of Healthcare Engineering, 2021.

[13] Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139, 106662.

[14] Goyal, S. (2022). Handling class-imbalance with KNN (neighborhood) under-sampling for software defect prediction. Artificial Intelligence Review, 55(3), 2023-2064.

[15] Jin, C. (2021). Software defect prediction model based on distance metric learning. Soft Computing, 25, 447-461.

[16] Zhu, K., Ying, S., Zhang, N., & Zhu, D. (2021). Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. Journal of Systems and Software, 180, 111026.

[17] Khurma, R. A., Alsawalqah, H., Aljarah, I., Elaziz, M. A., & Damaševičius, R. (2021). An enhanced evolutionary software defect prediction method using island moth flame optimization. Mathematics, 9(15), 1722.

[18] Feng, S., Keung, J., Yu, X., Xiao, Y., Bennin, K. E., Kabir, M. A., & Zhang, M. (2021). COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction. Information and Software Technology, 129, 106432.

[19] Tang, S., Huang, S., Zheng, C., Liu, E., Zong, C., & Ding, Y. (2021). A novel cross-project software defect prediction algorithm based on transfer learning. Tsinghua Science and Technology, 27(1), 41-57.

[20] Yang, L., Li, Z., Wang, D., Miao, H., & Wang, Z. (2021). Software defects prediction based on hybrid particle swarm optimization and sparrow search algorithm. Ieee Access, 9, 60865-60879.

[21] Nassif, A. B., Talib, M. A., Azzeh, M., Alzaabi, S., Khanfar, R., Kharsa, R., & Angelis, L. (2023). Software defect prediction using learning to rank approach. *Scientific Reports*, *13*(1), 18885.

[22] O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[24] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.