# RNN Based Deep Learning Software Defect Prediction with Comparison to Ensemble Machine Learning

1st Abul Bashar Khan *23-92815-1*
2nd Anannya Rahman *23-92995-2*
3rd Md. Pial Hassan *23-92900-1*
4th Sabiba Tasnim Sameka *23-93092-3*

***Abstract*—Software testing for defects has been a topic for researchers alongside the modern practices of the software development lifecycle. The discipline of software research has more active research in the area of software testing. In this paper we have discussed using deep learning algorithms and machine learning approaches with proper data set (SFP XP-TDD) for software defect prediction. The data set has data from three developed software projects. Few classifiers were used in this study and then their data set was trained and tested with Rotation Forest classifier ensemble versions. We also have discussed about few research articles for defect predictions using machine learning algorithms focusing on deep learning. Some of the researchers have preferred ensemble ML algorithms for software defect predictions, while others suggested to use of feature selection algorithms for the improvement of the performance of classification. Deep learning algorithms are prone to failure if the dataset consists of a small sample size. In our research, we have demonstrated that the deep learning approach can outperform the machine learning approach for software defect prediction for a large data set. Our focus is to provide the outcome of the experiment of the ML model (RNN). A few other datasets such as Apache Active MQ and Eclipse were used for validating the efficiency of the method that we have proposed.**

***Index Terms*—recurrent neural networks (RNNs); ensemble machine learning; deep learning; software defect prediction**

## I. Introduction

Eliminating bugs and defects in software applications and products should be a priority for software developers as these issues can cause software products to fail. Engineers who can raise the overall quality of the software product, increase performance, and win over users' trust. However, during the software development life cycle, modifications to the software's functionality can result in brand-new flaws in a product that was previously error-free. Defect-inducing changes are defined as changes that cause defects to occur.

The paper [1] discusses that the software modules that have a higher probability to contain defects can apply various techniques for defect prediction and emphasizes its importance. The use of such techniques can help developers allocate limited resources more efficiently and effectively prevent software defects. The paper [1] also highlights the limitations of the existing defect prediction approaches, in which most cases software modules are classified either as non-faulty or faulty but do not provide a specific defect count. To address this limitation, the proposed approach utilizes deep learning techniques for the prediction of the defect count in software modules. It is observed that the proposed approach indicated in the paper [1] which is evaluated on software data sets of the real world outperforms state-of-the-art approaches. A more accurate defect prediction model can be resulted by using deep learning models which can achieve discriminative features from the available data in an automated process. The proposed approach makes predictions at the module level, which can help facilitate unit testing and defect inspection.

Overall, the proposed approach in the paper [1] can improve the quality of software and induce overall cost reduction for maintenance-related activities. The paper [2] discusses some of the problems with deep learning in machine learning techniques and proposes a possible solution.

The paper [3] discusses the problem with software defect density prediction by the use of deep learning and a possible solution. The paper [4] highlights the problems with existing software defect prediction solutions and also describes the utilization of deep learning and random forest techniques for conducting experiments. A total number of five dissimilar data sets were used for the experiment. The model suggested in the paper [4] achieved 90% accuracy by using 10-fold cross-validation in predicting defects.

The results demonstrate that deep learning and random forest techniques have more accuracy than the Bayes network and SVM on all of the five used data sets. Additionally, the findings suggest that deep learning can serve as a competitive classifier and offer more robust predictions for the detection of defects [4]. The section "Approach" discusses several approaches and the "Results" section discusses the results from the papers [1] [2] [3] [4]. The data sets such as Apache Active MQ, Eclipse, etc. are widely used datasets in software fault prediction-related studies and approaches. RNN which is based on deep learning is used for studying these data sets with the objective to predict software defects. This has proven to have higher rates of performance.

Moreover, a total number of five classifiers of machine learning were used. These classifiers are commonly used in the literature of the researchers and studies. The data sets were also executed with the classifier Rotation Forest (RF) and their ensemble versions. Finally, the obtained results of these two approaches were compared to evaluate the method's performance. Three different software using the XP, TDD software development model approach related to Accounts,

Employee Management, and Inventory [5] to offer the result for the prediction of software defect. A tool for managing records of software defects use was developed to record the software-related questions, defect records, etc. for about a year.

For prediction of software defect using DL and ML a dataset was created[5]. The data related to Chidamber and Kemerer (CK) metrics, max and mean depth, number of lines, statements, and errors, and Cyclomatic Complexity was taken from the software database.

## II. APPROACH

At first, for software defect prediction we have gone through the proposed [1] deep learning-based model. The approach [1] consists of two steps, the first step is to train a deep learning-based model for defect prediction and based on the trained model perform defect prediction for new modules. Mapping between a module and the number of defects predicted is used for defining the defect prediction model. The proposed approach uses deep learning techniques to find this type of mapping. The data prepossessing step involves applying natural logarithm transformation and also does data normalization to the software metrics that are extracted. For defect prediction, the deep learning neural network-based model has one input layer with two hidden layers and one output layer.

For each of the layers, the kernel initializer, activation function, input, and output dimensions are specified. The number of predicted defects in the software module is the output of the neural network. The proposed approach has been evaluated on real-world software data sets and has shown promising results. The approach proposed can potentially improve the quality of software and cause overall cost reduction for the activities related to maintenance.

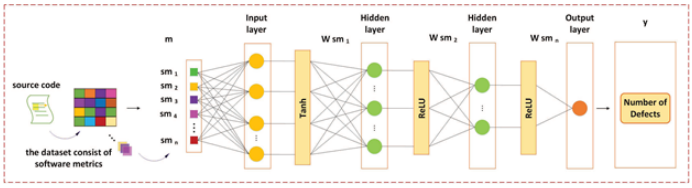


Fig. 1. Neural network overview.Adopted from [1]

The paper [2] discusses a few problems. Deep learning has become quite a popular approach in the field of software development, and it comes with certain downsides. Some common problems include:

1) Data scarcity: Deep learning models typically require large amounts of labeled training data, which can be challenging to obtain in certain domains or for specific tasks.
2) Interpretability: Due to the fact that deep learning models can often function as black boxes, which makes the understanding and interpretation of their decision-making processes difficult. This lack of interoperability can be a concern in domains where the ability to explain is vital.
3) Generalization: At times deep learning models might have difficulty to generalize well to new or unseen data if the training data differs significantly from the test data. Over-fitting, where the model becomes too specialized for the training data, is a common challenge.

According to the paper [2] Researchers and developers have proposed various solutions to address these challenges:

1) Data augmentation: Techniques such as image rotation, flipping, and noise addition can help increase the effective size of the training data set and improve the generalization capabilities of deep learning models.
2) Transfer learning: Pre-trained models trained on large-scale data sets can be fine-tuned on specific tasks with limited data, allowing the model to leverage knowledge learned from the broader domain.
3) Explainable AI: Researchers are actively looking for methods to make deep learning models more interpretable. Techniques such as attention mechanisms, visualization tools, and model introspection can help shed light on the decision-making process of these models.

This part included a brief description of the research goal and objectives along with methodologies [2]. To summarize the research implementation of The Deep Learning Algorithms in short DLA for the SDP. The authors of this work used the Kitchenham and Charters SLR technique. The four key tasks conducted by the authors were: (1) identifying the objective and the focused Questions (FQs), (2) choosing apposite primary studies, (3) Extraction of the collected sample data, and last but not least its (4) data synthesis along with data reporting. This study's scope and objectives were developed utilizing the "Goal-Question-Metric technique". Examining the state of software defect prediction, or SDP in short, with a focus on observation and in-depth analysis relating to the platforms of SDP, Machine learning categories, data sets, source code presentation etc. The objective of this study are to compile, analyze, and synthesize the body of information and data on the application of DL algorithms for SDP.

RQ1: – What are the SDP scenarios (WPDP, CPDP, HDP, etc.) were applied?
RQ2: – What are the different types of Machine Learning (ML) (categories such as supervised/unsupervised/semi-supervised learning) that have been applied to this DL-based SDP research?
RQ3: – Which public data sets were used in this research work to create and evaluate the ML/DL model for SDP?
RQ4: – By following which method did researchers of this research work created the Deep Learning Model in short known as DLM of SDP from the program source code?
RQ5: – At which stages of granularity level did the researchers try to perform the SDP?
RQ6: – What kind of strategies did researchers try their best to use to address the class imbalance problem (in short CiP) in SDP?

RQ7: – What kind of DL algorithm (such as CNN, LSTM, etc.) was tried to apply?
RQ8: – What are the different kinds of evaluation metrics along with validation system approaches that were used?
RQ9: – How often have researchers provided replication tools that support the DL model of SDP's replication functionality?
RQ10: – What are the difficulties in using the DL for SDP and what solutions have been proposed?

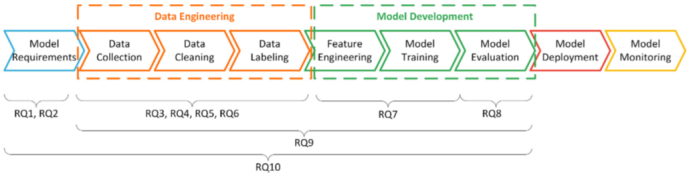


Fig. 2. RQs mapped to the ML model life cycle. Adopted from [2]

The RQs are mapped to the proposed ML model life cycle by Amershi et al. (2019) For the sake of simplicity, the feedback loops and iterations throughout the life cycle were left out. Researchers choose the SDP scenario(s) they will concentrate on during the model requirements stage and choose the best ML category for the SDP challenge. Teams search for existing data sets or create new ones throughout the data engineering stages.

The problem addressed in the paper [3] is the challenge of predicting defect density in software development, this is crucial for providing clients with a trustworthy and high-quality software system. The study suggests utilizing deep learning to create a defect density prediction model that can aid in the identification of flawed modules, lower testing expenses, and boost resource efficiency. The defect density data set's data sparsity, which might skew the final forecast, is one of the primary issues addressed in the research. In order to address this intrinsic problem, deep learning is suggested in the study. It is also said that deep learning has shown success when dealing with sparse data. The two research issues surrounding the prediction of software fault density are the focus of this work [3]. The effectiveness of deep learning algorithms is contrasted with that of conventional machine learning methods in the first question. The authors [3] use 28 publicly available data sets to evaluate k-Nearest Neighbors, Support Vector Regression, Random Forest, Naive Bayes, and many Linear Regression with an improved version of the conventional GRNN that includes many dense layers. The Wilcoxon signed-rank test is used to statistically compare the data, and it reveals that the suggested deep GRNN model performs noticeably better than other comparable machine learning models. The second research topic [3] examines how data sparsity affects deep learning-based defect density prediction. The authors use the sparsity ratio, which determines the ratio of non-zero faulty modules to zero defective modules, to categorize the data sets into four groups. Following that, the effectiveness of the proposed deep GRNN model and other machine learning methods is evaluated across each set of data sets. Overall findings [3] show that the suggested model performs better than the majority of machine learning models, especially on extremely high and high sparsity data sets.

The paper [4] reviews recent research on just-in-time software fault prediction and the advantages of deep learning in software engineering. The approach used in experiments is presented, and the experimental analysis and evaluation results of the proposed model are reported. The error level in accuracy can be reduced, and false predictions can be avoided as the data grows by the model.

The evaluation approach's general process is demonstrated. The defect data is split into a training set and a test set. In a supervised method, the training set is used to learn the distribution of classes and how they relate to other attributes. An algorithm or a series of steps can be used to anticipate additional examples of errors once the model has been learned. Testing the model on hypothetical test data and recording the results allows for the measurement of the model's accuracy. The procedure is then repeated in order to improve the model's accuracy using a new batch of training data. The performance of the model and its comparison with other classifiers is evaluated using the 10-fold cross-validation technique, which splits the data into 90

The data from the CM1 repository originates from a NASA spacecraft instrument written in "C". McCabe and Halstead feature extractors, designed to objectively characterize code features associated with software quality, were used to extract the data. These measures are module-based, with a module being the smallest unit of functionality in the code (i.e., a function or method in C).

The data from the JM1 repository originates from a real-time predictive ground system written in "C". The KC1 repository contains data from a "C++" system that is responsible for managing storage for receiving and processing ground data. The KC2 repository contains data from C++ functions utilized for science data processing. It is a separate project from KC1 and is operated by different personnel, but shares some third-party software libraries with KC1. Lastly, the PC1 repository contains data from C functions used in flight software for an earth-orbiting satellite. To evaluate the performance of different approaches on data sets CM1, JM1, and KC1, Precision, Recall, and F-measure were used. The precision was calculated at

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

Where True Positive and False Positive is denoted as TP and FP respectively.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

Where False Negative is denoted as FN.

To calculate the F-Measure both recall and precision are considered:

$$\text{Fmeasure} = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall} \tag{3}$$

The F-measure [1] was chosen because of the unbalanced classes in the data sets. To make comparative talks simpler, the F-measure was supplied for each strategy. Among all methods, the Bayes network obtained the lowest F-measure (0.718 for CM1, 0.71 for JM1, and 0.73 for KC1 data sets). With F-measures of 0.854 and 0.852 for CM1, 0.787 and 0.722 for JM1, and 0.848 and 0.786 for KC1, respectively, the Random forest and SVM beat the Bayes network. With an F-measure of 0.861 for CM1, 0.755 for JM1, and an unreported value for KC1, the deep learning strategy achieved the greatest F-measure. The Deep Learning technique exhibited a greater rate of correctly identifying the defect and non-defect instances when compared to the Bayes network, but it did not perform noticeably better than Random Forest and SVM for the JM1 and KC1 data sets.

Deep Learning: A set of machine learning algorithms known as "deep learning" gradually extracts higher-level information from the raw input by using many layers [6].

Software defect prediction: Software defect prediction (SDP) is a procedure for predicting whether software entities are defective. It's widely acknowledged that addressing defects throughout the development phase saves costs compared to repairing them after release [6].

LSTM: Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN). LSTMs can handle long term dependencies; this is the ability of LSTMs. LSTMs are designed for sequence prediction problems or recognizing patterns to solve these problems [6].

Applications of LSTMs:

1) Speech Recognition
2) Natural Language Processing (NLP)
3) Machine Translation
4) Time Series Forecasting
5) Software defect Prediction and so on

In RNN, as the number of inputs raised, there is a possibility that the RNN could give a wrong output. The concept of LSTM is introduced to address this issue [6].

The key of LSTM is the cell state. For long sequence data, RNN can't predict the exact output. To address this, in LSTMs, the cell/memory state is introduced. LSTMs also include sigmoid layers that provide output, with the sigmoid function having values of either "0" or "1". "0" means let nothing through and "1" means let everything through. LSTMs have three additional states used to protect and control the cell/memory states [6].

Forget Gate: The forget gate prioritizes the importance of the data in the cell state. It calculates the importance and communicates the decision to the cell state using the formula:

$$ft = \sigma(Wf \cdot [ht-1, Xt] + bf) \tag{4}$$

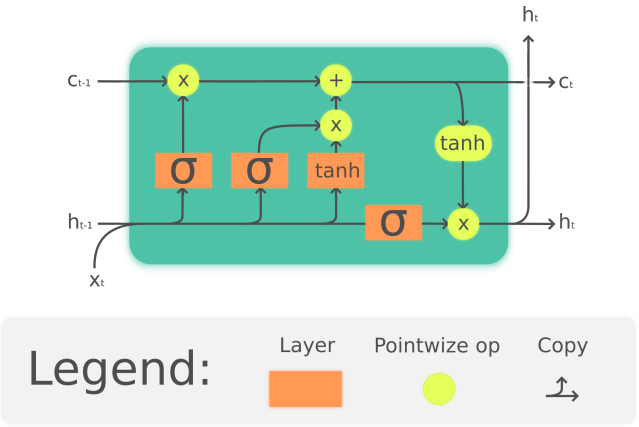


Fig. 3. The Long Short-Term Memory (LSTM) architecture

Input gate: It consists of two parts, a sigmoid function for determining what types of data to store and a tanh function to update the values. The formulas for input gate are:

$$it = \sigma(Wi \cdot [ht-1, Xt] + bi) \tag{5}$$

$$\sim Ct = \tanh(WC \cdot [ht-1, Xt] + bC) \tag{6}$$

Output gate: It contains a sigmoid and a tanh function to decide the output. The formulas are:

$$ot = \sigma(Wo \cdot [ht-1, Xt + bo]) \tag{7}$$

$$ht = ot \times \tanh(Ct) \tag{8}$$

In the context of software defect prediction, LSTMs can be used to analyze and predict software defects or bugs based on historical data. LSTMs, with their ability to capture long-term dependencies in sequential data, can be a powerful tool for software defect prediction. However, it's essential to have a good understanding of the software development process and dataset to effectively apply LSTM models for this task [6].

Here's how LSTMs can be applied to software defect prediction:

**Data Preparation:** Gather information about previous software development, such as bug reports, code modifications, and other pertinent data. Assemble the data into a sequence that captures the chronological order of occurrences, with each data point denoting a time step [6].

**Feature Engineering:** Extract specific features from the data, such as code complexity metrics, developer experience, project size, code churn, and defect history [6].

**Data Encoding:** Transform the data into an LSTM-input format by encoding it using techniques like one-hot encoding or word embeddings to convert text-based data or categorical variables into numerical representations [6].

**Training:** Sort the data into sets for validation and training the model. During training, optimization techniques like Adam or backpropagation are used to update the weights of the LSTM [6].

**Continuous Improvement:** Retrain the model with each new set of data collected to help it adjust to evolving development patterns and increase forecast accuracy over time [6].

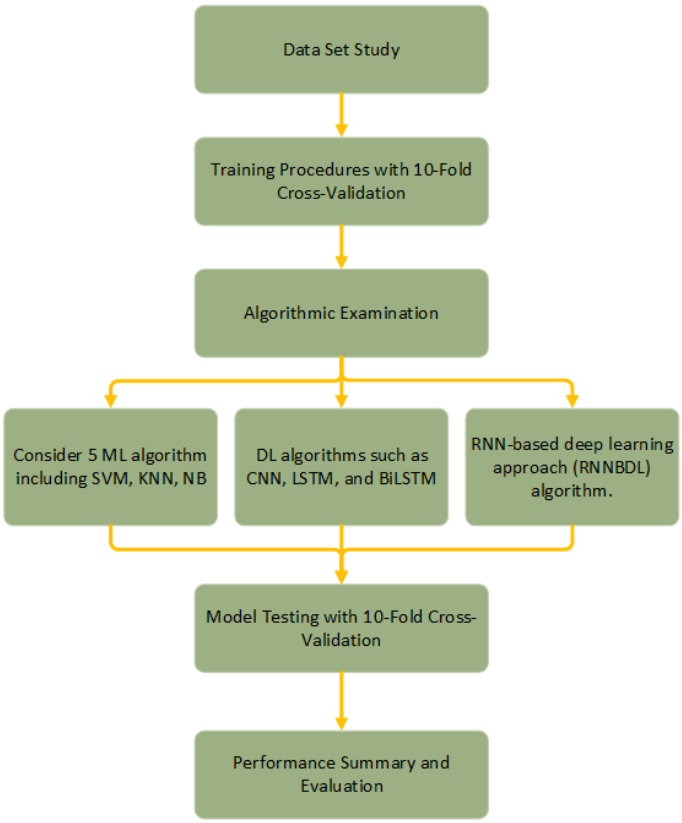


Fig. 4. Planned Methodology

## REFERENCES


[1] Lei Qiao, Xuesong Li, Qasim Umer, Ping Guo, Deep learning based software defect prediction, Neurocomputing, Volume 385, 2020, Pages 100-110, ISSN 0925-2312, https://doi.org/10.1016/j.neucom.2019.11.067.

[2] Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, Bedir Tekinerdogan, On the use of deep learning in software defect prediction, Journal of Systems and Software, Volume 195, 2023, 111537, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2022.111537.

[3] F. Alghanim, M. Azzeh, A. El-Hassan and H. Qattous, "Software Defect Density Prediction Using Deep Learning," in IEEE Access, vol. 10, pp. 114629-114641, 2022, doi: 10.1109/ACCESS.2022.3217480.

[4] R. U. Khan* et al., "Software Defect Prediction Via Deep Learning," International Journal of Innovative Technology and Exploring Engineering, vol. 9, no. 5. Blue Eyes Intelligence Engineering and Sciences Engineering and Sciences Publication - BEIESP, pp. 343–349, Mar. 30, 2020. doi: 10.35940/ijitee.d1858.039520

[5] Borandag, E. Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques. Appl. Sci. 2023, 13, 1639. https://doi.org/10.3390/app13031639

[6] L. L. S. Q. Jiehan Deng, *"Software defect prediction via LSTM,"* The Institution of Engineering and Technology 2020, vol. 14, no. 4, pp. 443-450, 01 August 2020.