JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

Enhancing Prediction Reliability of Software Failure Using Recurrent Neural Networks

Okoro Christian Chukwuka[a], Abdulsalam, Ya'u Gital, Imam Mustapha, Lawal Abdulrahman, Patrick Fatima, Patrick Obed, Shima Stephen
Department of Mathematical Sciences,
Abubakar Tafawa Balewa University Bauchi, Bauchi State, Nigeria

ABSTRACT
The prediction of software failures is done by using the historical failures collected previously when they occur. Software failure is said to occur when the software runs in an operational profile. Controlling failures in software require that one can predict problems early enough to take preventive measure. Predicting defective code in the software development process is a crucial aspect of software analytics. Recently, a number of computational approaches were employed to solve this task, modern software is developed with more sizes and functions, and assessing software defects is a remarkably difficult task. These approaches have significantly facilitated the prediction of software defects; however, their performances e.g., mean square error (MSE) and squared correlation coefficient (R2) requires significant improvements. The recently developed deep learning model such as LSTM has shown to be suitable for good performance. This deep learning model do not only deepens the layer levels but can as well adapt to capture the training characteristics. A comprehensive, in-depth study ultimately shows the model can have suitable performance. Thus, this research work proposed a deep learning approach, specifically the novel deep recurrent neural network (MLP-LSTM) in order to enhance the performance of software defect prediction on five benchmark datasets obtained from NASA project available at PROMISE repository portal. We evaluate the performance of the proposed model against the state-of-the-art approach using mean square error (MSE) and error correlation (R2). Simulation results show that the proposed approach achieved an excellent performance in terms of MSE as against the existing work on the benchmark KC2 and MIS dataset. In addition, the proposed approach outperformed the existing approaches in all cases of the software defect datasets (CM1, JM1, KC1, KC2 and PC1) that were used for evaluating the experiment both in terms of MSE and R2.

ARTICLE INFO
Article History
Received: March, 2023
Received in revised form: April, 2023
Accepted: May, 2023
Published online: June, 2023

KEYWORDS
Prediction, Software, Recurrent Neural Networks

INTRODUCTION

Software defect prediction is a process that uses a model to predict the code areas that potentially contain defects (Qiao et al., 2020). Automated software defect prediction is a process where classification and/or regression algorithms are used to predict the presence of non-syntactic implementation errors (henceforth; defects) in software source code (Gray et al., 2011). To make these predictions, such algorithms try to generalize upon software fault data; observations of software product and/or process metrics coupled with a level of defectiveness value. This value typically takes the form of a number of faults reported metric, for a particular software unit after a certain amount of time (post either code development or system deployment)(Gray et al., 2011). Software defect prediction (SDP) is the most dynamic research area in software engineering. SDP is a process used to predict the deformities in the software (Sethi, 2016). To identify the defects before the arrival of item or aimed the software improvement, to make software dependable, defect prediction model is utilized. It is always appropriate to predict the

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

defects at early stages of life cycle. Hence to predict the defects before testing the SDP is done at end of each phase of SDLC (Sethi, 2016).

Software failure occurs when the software runs in an operational profile. Controlling failures in software require that we can predict problems early enough to take preventive action. The prediction of software failures is done by using the historical failures collected previously when they occur (Benaddy, El Habil, El Meslouhi, & Krit, 2018). Predicting defective code in the software development process is a crucial aspect of software analytics. Software testing company Tricentis estimated the cost of software bugs at $1.1 trillion in 2016. Quality assurance resources are usually limited to sustaining the software. Predicting faulty units exactly allows developers and managers to prioritize their actions in the software development cycle and to address these faults. In defective software, a faulty unit might result from various factors that are hard to detect using human processes such as code review (Li, Lessmann, & Baesens, 2019). Given the large size, number of lines of code, and complexity of a typical software project, there is a need for a much more scalable approach. Many researchers (Alhazmi & Malaiya, 2005; Fenton, Neil, & Marquez, 2008) have proposed quantitative approaches to this research problem. Thanks to the proliferation of IT artifacts, researchers can use increasing amounts of data and conduct a wide range of experiments to pursue a better solution (Li et al., 2019).

Different software reliability models, such as parameter and non-parameter models, have been developed in the past four decades to assess software reliability in the software testing process. Researchers have developed many parameter models in the past 40 years to evaluate software reliability, such as those based on non-homogeneous Poisson process, stochastic differential equation, and Bayes process. Parameter models must generally estimate external parameters by utilizing least squares or maximum likelihood estimation (J. Wang & Zhang, 2018). Thus, different estimation methods can provide different external parameter estimation results. These methods also require assumptions established in advance to build the relevant parameter models. Several of these assumptions are reliable with the realistic testing process, whereas others are not in the realistic testing scenarios. Thus, the assumptions adopted in a testing case must be different from those in other testing cases because of the complexity of software testing (Wang & Zhang, 2018). Furthermore, proposing the assumptions to include all testing circumstances is practically difficult (Wang & Zhang, 2018). The assumptions proposed in parameter models are subjective, limited, and have incomplete testing feature selection to simplify and facilitate modelling. Proposing objective and realistic assumptions in parameter models remains a challenging task (Wang & Zhang, 2018).

Although these models can effectively assess software reliability in certain testing scenarios (Wang & Zhang, 2018), no single model can accurately predict the fault number in software in all testing conditions. In particular, modern software is developed with more sizes and functions, and assessing software reliability is a remarkably difficult task (Wang & Zhang, 2018).

The process of software defect prediction includes three main steps: collecting a historical defect dataset; using the historical data to train a regression or classification model using machine learning or deep learning techniques; and applying the trained model to predict the number or probability of software defects (Kalaivani & Beena, 2018). From the perspective of different dataset granularities, software defect prediction techniques can be divided into four categories (Kalaivani & Beena, 2018): package level (Schröter, Zimmermann, & Zeller, 2006), file (modules) level (Yan, Fang, Lo, Xia, & Zhang, 2017), method level (Hata, Mizuno, & Kikuno, 2012), and change level (Kamei et al., 2012). From the perspective of different metrics, it can be broken down into two defect prediction categories: static and dynamic. Static defect prediction techniques mainly involve predicting the number of defects or the defect distribution using static software metrics. Dynamic defect prediction techniques mainly involve predicting the distribution of software defects over time using the defect generation time.

Most defect prediction techniques build defect prediction models that use traditional hand-crafted features, including

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

Halstead's software volume metrics and McCabe's cyclomatic complexity metrics. Such approaches are centred on statistical and machine learning theories. These approaches, which include support vector regression (SVR) (Qiao et al., 2020), fuzzy support vector regression (FSVR) (Qiao et al., 2020) random forest (RF) (Sun et al., 2018), and Adaboost (Ren, Qin, Ma, & Luo, 2014) models, first build a regression or classification model and then use the model to predict the number or probability of defects for a given unit of source code. These approaches have significantly facilitated the prediction of software defects; however, their performances (e.g., mean square error and squared correlation coefficient) requires significant improvements (Qiao et al., 2020). The recently developed deep learning model, called deep neural network (NN) model, has appropriate prediction performance. This deep learning model not only deepens the layer levels but can also adapt to capture the training characteristics (J. Wang & Zhang, 2018).

Based on the literature, we have seen that regression and classification models are used to predict the number or probability of defects for a given unit of source code. These approaches have significantly facilitated the prediction of software defects; however, their performances (e.g., mean square error and squared correlation coefficient) requires significant improvements (Qiao et al., 2020). In the software defect prediction field, additional new and commercial datasets are needed to ensure reliability and generalization, thus, investigating new performance criteria for defect prediction models would also be interesting and critical to ensuring more generalized, robust and reliable prediction techniques (Qiao et al., 2020). (Benaddy et al., 2018; Dam et al., 2018; Wang & Zhang, 2018) opined that deep NN model base on recurrent neural network (RNN) has lower prediction errors than NN models.

Thus, this research work proposed a deep learning approach, specifically the deep recurrent neural network Long Short Time Memory (LSTM) in order to enhance the prediction performance of software fault using deep learning algorithms using five benchmark datasets obtained from NASA project available at PROMISE repository (http://promise.site.uottawa.ca/SERepository/) (Karim, Warnars, Gaol, Abdurachman, & Soewito, 2017).

Advantage and Motivation

Software systems have become larger and more complex than ever. Such characteristics make it become very challengeable to prevent software defects. Therefore, automatically predicting the number of defects in software modules is essential and may help developers efficiently to allocate limited resources. Predicting the defects before testing the SDP is done at end of each phase of SDLC. It helps to reduce the cost as well as time and hence aids in producing high quality software. Moreover, accurately predicting faulty software units helps practitioners target faulty units and prioritize their efforts to maintain software quality.

LITERATURE REVIEW

In this section, we introduce related works on software metrics, classification and regression models for software defect prediction, and deep learning and its applications.

Software Metrics

Software metrics are measures of the success of a software process (Yang et al., 2016). In theory, metrics can help to improve the development process and provide companies with the information that makes future projects more predictable and efficient. A number of software defect prediction techniques that makes use of software metrics have been proposed. This can be divided into two types: supervised defect prediction approaches and unsupervised defect prediction approaches. Supervised defect prediction approaches use historical datasets to train a defect prediction model for example, (Benaddy et al., 2018; Sun et al., 2018; Wang & Zhang, 2018) used historical data from Samsung and NASA software fault metric datasets to predict the reliability of the respective project. Unsupervised defect prediction approaches can predict defect proneness without requiring a defect dataset (Yang et al., 2016). That approach can be used when a training dataset is insufficient or is not available. For example (Yang et al., 2016) proposed an unsupervised approach that ranks the change metrics in

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

descending order based on the reciprocal of the raw value of each change metric. A typical NASA software metric data is presented in table 1 below:

Table 1: A typical NASA software metric data (Qiao et al., 2020)

Metric Type	Software Metrics	*Description*
McCabe	LOC	McCabe's code line count
	V(G)	McCabe's "cyclomatic complexity"
	EV(G)	McCabe's "essential complexity"
	IV(G)	McCabe "design complexity"
Halstead	N	Halstead's total operators + operands
	V	Halstead's "volume" for each module
	L	Halstead's program length
	D	Halstead's "difficulty"
	I	Halstead's "intelligence"
	E	Halstead's "effort"
	B	Halstead
	T	Halstead's time estimator
Line Count	LOCode	Halstead's line count
	LOComment	Halstead's count of blank lines
	LOBlank	Halstead's count of blank lines
	LOCode And Comment	Halstead's code and comment count for each module
Operator Operand	Uniq_Op	Unique operators
	Uniq_Opnd	Unique operands
	Total_Op	Total operators
	Total_Opnd	Total operands
Branch	BranchCount	of the flow graph

Regression Models for Software Defect Prediction

Fuzzy logic

Fuzzy logic has been successfully used to solve variety of problems in system identification, signal processing and control. A fuzzy model structure can be represented via a set of fuzzy If-Then rules. A fuzzy rule is divided into two parts, the antecedent and the consequence. The antecedent variables rules reflect information about the process operating conditions, while the consequent of the rule is usually a linear regression model, which is valid around the given operating condition. For example (Aljahdali & Sheta, 2011) explore the use of fuzzy logic to build a SRGM make analysis on John Musa of Bell Telephone Laboratories data set. They result provided show the potential advantages of using fuzzy logic in solving this problem (Aljahdali & Sheta, 2011).

Classification and Prediction Models for Software Defect Prediction

Regression analysis is a form of predictive modelling technique, which investigates the relationship between a dependent (target) and independent variable (s) (predictor) (Qiao et al., 2020). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables (Qiao et al., 2020). Whereas classification models have been widely studied in defect prediction, research on defect prediction using regression models is more limited (Qiao et al., 2020). A regression model considers defect prediction as a ranking task, where the software modules are ranked according to the number of predicted defects they contain. Predicting the number of defects explicitly in software modules is not an easy task. Therefore, an accurate regression model is necessary (Qiao et al., 2020).

Recurrent Neural Networks LSTM

Deep recurrent neural network originates from recurrent neural network (RNN), which is a class of artificial neural network where connections between nodes create a directed graph (Chung, Gulcehre, Cho, & Bengio, 2014; Cui, Henrickson, Ke, & Wang,

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

2019). It models the temporal dynamic behaviors exhibited in time series data through the use of feedback connections to recall the neural states at previous time steps. A typical structure of an RNN is shown in Fig. 2. Unlike feedforward neural networks, RNN is capable to use the neural internal states to process time series sequences of inputs, making them appropriate for renewable energy forecasting (Hu & Chen, 2018). A RNN can be extended to a deep RNN via different ways as shown by (Pascanu, Gulcehre, Cho, & Bengio, 2013), there are four different ways to formulate deep RNN from conventional RNN. The first method is to learn more non-temporal structure from the inputs by deepening the input-to-hidden function. It is establish that this formulation tends to better flatten the manifolds near which the data concentrates and disentangle the underlying variation factors than the original inputs. The concept of depth in RNN was initially argued (Gulcehre, Cho, Pascanu, & Bengio, 2014), but base on the architecture of RNN, three areas can be making deeper in other to construct a DRNN, this includes; hidden to hidden function, hidden to hidden transition and hidden to output function (Gulcehre et al., 2014).

Various deep RNN models have been proposed (Gulcehre et al., 2014), However, deep RNNs may increase the computation complexity, especially when the time series data exhibits long tails (Soltani & Jiang, 2016; H. Wang, Lei, Zhang, Zhou, & Peng, 2019). One feasible solution has to do with adopting recurrent and convolutional operators for model development. Another possible solution attributes to the use of bidirectional calculations that can capture the impacts of both past and future states (H. Wang et al., 2019). Deep RNN can have other stored state, which are under direct control by the neural network. Also, the stored states can also be substituted by another neural network with time delays or feedback loops. Such controlled storages are the basis of long short-term memory network and gated recurrent units. They both have distinctive temporal dynamic behavior and can mitigate the exploding and vanishing gradient problems. Therefore, long short-term memory and gated recurrent network exhibit promising performance in the context of forecasting (H. Wang et al., 2019).

.

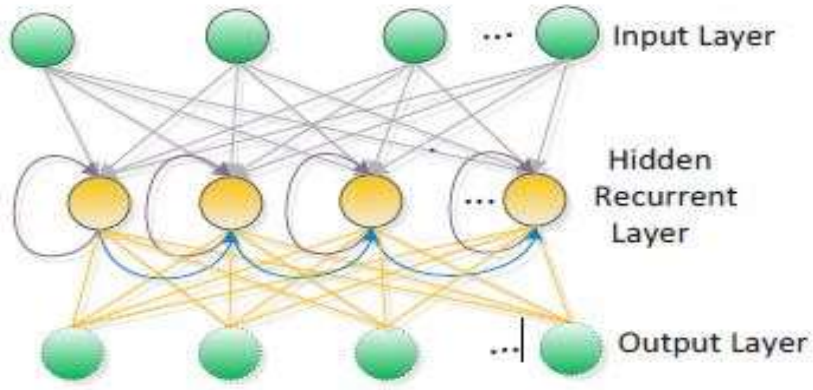


Figure 1: A typical structure of a recurrent neural network(H. Wang et al., 2019)

RELATED WORK

The process of software defect prediction includes three main steps: collecting a historical defect dataset; using the historical data to train a regression or classification model using machine learning or deep learning techniques; and applying the trained model to predict the number or probability of software defects (Kalaivani & Beena, 2018). From the perspective of different dataset granularities, software defect prediction techniques can be divided into four categories: package level (Schröter et al., 2006), file (modules) level (Yan et al., 2017), method level (Hata et al., 2012),

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

and change level (Kamei et al., 2012). From the perspective of different metrics, it can be broken down into two defect prediction categories: static and dynamic. Static defect prediction techniques mainly involve predicting the number of defects or the defect distribution using static software metrics. Dynamic defect prediction techniques mainly has to do with predicting the distribution of system defects over time using the defect generation time. Most defect prediction techniques build defect prediction models that use traditional hand-crafted features, including Halstead's software volume metrics and McCabe's cyclomatic complexity metrics. These approaches are based on statistical and machine learning theories. These approaches, which include support vector regression (SVR) (Qiao et al., 2020), fuzzy support vector regression (FSVR) (Qiao et al., 2020) random forest (RF) (Sun et al., 2018), and Adaboost (Ren et al., 2014) models, first build a regression or classification model and then use the model to predict the number or probability of defects for a given unit of source code. These approaches have significantly facilitated the prediction of software defects; however, their performances (e.g., mean square error and squared correlation coefficient) requires significant improvements (Qiao et al., 2020).

The recently developed deep learning model, called deep neural network (NN) model, has suitable prediction performance (Wang & Zhang, 2018). This deep learning model not only deepens the layer levels but can also adapt to capture the training characteristics. A comprehensive, in-depth study and feature excavation ultimately shows the model can have suitable prediction performance (Wang & Zhang, 2018). For example, (Tamura, Matsumoto, & Yamada, 2016) Proposed Software Reliability Model Selection Based on Deep Learning NN, the method based on the deep learning proofs it can assess better than the method based on traditional neural network.

Similarly, (Wang & Zhang, 2018) Proposed a Deep Learning Model based on the RNN Encoder Decoder for Software Reliability Prediction and evaluated it against a traditional neural network, findings from the research (Wang & Zhang, 2018) show that the proposed deep NN model has lower prediction errors than NN models.

The work in (Dam et al., 2018) applied tree-structured LSTM network (Tree-LSTM) model for software defect prediction and evaluated the results against Random Forests and Logistic Regression. The experimental results demonstrate Promising results. The major limitation of the study is that the study only focus on software defect, from the evaluation, the approach can be applied into practice to further extend the prediction to specific types of defects such as security vulnerability and safety-critical hazards in code.

Furthermore, (Sun et al., 2018) utilize Deep Architecture Networks of Variable Auto Encoder in Software Fault Prediction and evaluate it against Support Vector Machine, Linear Regression, Random Forest and Decision Tree. The Variable Auto Encoder method has shown better ability to predict fault of software than the other method. More so, this study Lacks Generalization due to fewer datasets employ for evaluation.

Similarly, (Qiao et al., 2020) Proposed Deep neural network Based Software Defect Prediction and evaluated it with other regression models including SVR, FSVR AND DTR. The proposed method significantly reduces the mean square error by more than 14% and increases the squared correlation coefficient by more than 8% in the defect prediction field, but one of the major limitations of the work is insufficient data input, thus, additional new and commercial datasets are needed to ensure more reliability and generalization.

The study in (Ferenc, Bán, Grósz, & Gyimóthy, 2020) shows that deep learning with static metrics can indeed boost prediction accuracies thus, the study applies Deep learning in static, metric-based bug prediction and evaluate the prediction accuracy against Forest, SVM, KNN and Ensemble.

METHODOLOGY

As mentioned in the previous section both feed-forward and recurrent neural networks are applied successfully in software failure prediction. The recurrent architectures have its advantage to share information over the time, from one-time step to the next one. This kind of neural networks is suitable for predicting cumulative failure as the present failure is summed with the past one. In this research

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

work, we adopt an LSTM recurrent neural network due to its abilities to model complex and dynamic data with high accuracy requiring fewer parameters. They results obtained will be evaluated against the existing study of (Qiao et al., 2020), thus this can be achieved by collecting and reusing the same datasets (NASA Software Defect Dataset) use in (Qiao et al., 2020) from PROMISE repository Portal containing the software defect metrics (Gray et al., 2011; Qiao et al., 2020) and conducting some preprocessing operations (data normalization) on the given software metrics. Secondly is to train a specially designed recurrent-based deep learning neural network model to predict the number of defects in code input the modeled data to the trained regression model to predict the number of defects in each provided module.

This research work proposed a deep learning approach, specifically the novel deep recurrent neural network (LSTM) to prediction software fault Five (5) commercial datasets obtained from NASA project available at PROMISE repository portal which is an addition compare to the two (2) datasets that was used in the existing work of (Qiao et al., 2020). Figure 2 illustrates the flowchart of the proposed framework. Primarily, the first step of the framework involves the data set pre-processing stage.

Data set Pre-processing Stage

The first part of the entire strategy is the dataset stage, which reads all the defined dataset. Different operations are executed on those datasets including connecting to the database that includes dataset loading and reading files. Data normalization is a commonly used technique that transforms large data value ranges into small range values (or binary values). Notably, we perform min-max normalization because it is a commonly used normalization approach due to its high accuracy and high learning speed. Furthermore, min-max normalization does not change the dataset distribution.

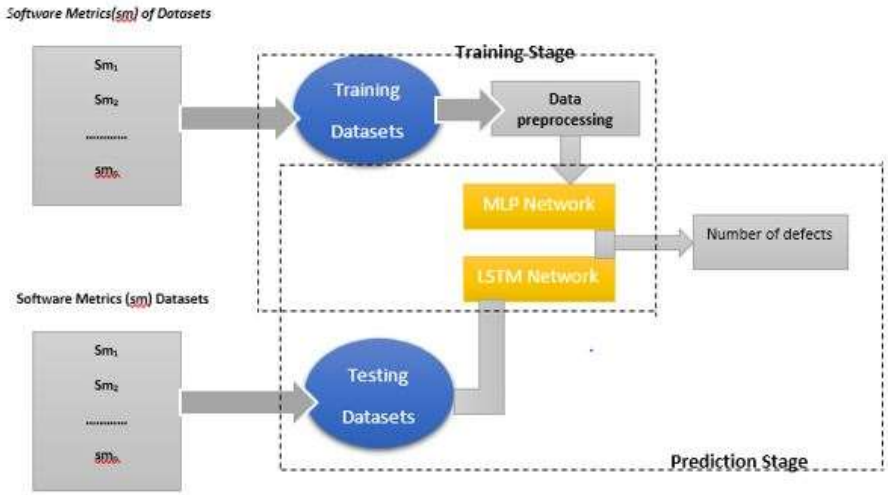


Fig 2: Architecture for the implementation of the proposed (MLP-LSTM)

Training Stage

In this research work, we adopt Bayesian Back Propagation as the training algorithm, the normalized data x= {x_1, x_2, ..., x_n}, are used to train the RNN to predict the future cumulative failures. To do so the data sequence should be grouped into vectors of length w (sliding window) that can be denoted as x= {x_{k-w}, x_{k-w+1}, ..., x_{k-1}}, k = w, w+1, ..., n for any k the vectors contain w inputs. The target data y= {y_1,

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

y_2, ..., y_n} contains the same amount of data in input data. The two training data sets map the relationships of the network. The training data sets x and y are used to train the network for adjusting its weights W, U and V and biases b and c. The weights and biases are randomly initialized within the interval [-0.1,0.1]. The network would then be optimized using ADAM optimization algorithm.

Deep Learning Prediction Stage

The prediction process is divided into two stages; the training stage and the testing stage. At the training stage the network is trained with 70% of available collected data. The testing stage is performed after the training by using 30% of the collected data. The process is repeated a number of training epochs. At each epoch step the network is retrained for new prediction and testing for each data inputs within the project. The LSTM deep recurrent neural was adopted for the prediction, this kind of neural networks is suitable for predicting cumulative failure as the present failure is summed with the past one. In this research we adopted a simple MLP neural network to build on the top layer of the LSTM network. The MLP here will extract and generate the initial forecast data in which the LSTM network will input it and used the data to predict the number of software defect in each of the datasets.

Experimental Setup

The developed algorithm presented in Figure 2 were implemented using MATLAB R2018a. The computer used is Dell laptop running on Windows 10 Operating System with 8GB RAM, 1 TB HDD and Pentium ® Core i7 processor. For all experiments, we used MATLAB R2018a. The datasets are divided into 70% for training and 30% for testing using stratified sampling in order to preserve class distribution as much as possible. Furthermore, to eliminate the effect of features that have different scales, all datasets are normalized using min–max normalization. All experiments are executed for five different runs, and each run includes 50, 100, 150, 200, 250 iterations respectively. For five (5) different software defect datasets, the simulation was repeated for 250 iterations in each case.

Dataset Description

We reuse the software metrics extracted from the KC1 dataset and the KC2 dataset which are the most commonly used software metrics for defect prediction. In order to ensure further reliability and accuracy of our results against the existing work, we further add three more commercial dataset which are, CMI, JMI and PC1. This is a PROMISE Software Engineering Repository data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering. As presented in table 1. The dataset includes five dimensions, i.e., McCabe, Halstead, Line Count, Operator and Operand, and Branch. We made used of 21 software metrics of the KC2 dataset (e.g., LOC, V(G), EV(G), etc.)

Choice of Metrics

The proposed model is evaluated on the first 70% of the modules and the last 30% of the modules. In this research we would use the number of defects for evaluation and calculation of the mean squared error (MSE) and the error auto correlation (R^2) to evaluate the performance of the defect prediction model. The number of defects is the number of defects contained in a software module. For a given module, our defect prediction model and other state-of the- art approaches can predict the number of defects. The closer the number of predicted defects is to the actual number of defects contained in a module, the better the performance of the defect prediction model.

The MSE measures the average of the squared errors, the squared difference between the estimated parameter and the true value of the parameter. The MSE measures the quality of a prediction model and better values are non-negative and closer to zero. Moreover, a smaller MSE represents better prediction model performance. Moreover, a smaller MSE means that the prediction model is closer to the optimal model. The MSE can be computed as follows:

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

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

where y_i represents the vector of actual values of the variable, $\acute{y}_i$ represents the predicted variable values, and n is the number of data points.

The R^2 in equation (1) measures the proportion of the variance in the dependent variable that is predictable from the independent variables, and it is calculated by squaring the correlation coefficient between the observed and predicted values in a regression model. The R^2 represents the strength of the relationship between an independent and dependent variable(s), and it measures the performance of a model. Its values range from 0 to 1 but can never be less than zero nor greater than zero for any regression model. An R^2 value between 0 and 1 shows the extent to which the dependent variable is predictable, which represents how well the regression model fits the datasets. An R^2 value closer to 1 indicates that the model has achieved a good fit to this problem and represents the strength of the relationship between an independent and dependent variable(s). A larger R^2 value represents a better model fit. The R^2 can be computed as follows:

$$R^2 = \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \quad (2)$$

where y_i represents the observed values of the dependent variable, $\acute{y}$ is the mean value, and $\acute{y}_i$ is the predicted value.

RESULTS PRESENTATION AND ANALYSIS

The proposed Deep Leaning (MLP-LSTM) is evaluated on the first 70% of the modules and the last 30% of the modules. Notably, we use the number of defects for evaluation and calculate the mean squared error (denoted as MSE) and the coefficient of determination (denoted as R^2) to evaluate the performance of the defect prediction model. The number of defects is the number of defects contained in a software module.

Performance Comparison against Benchmark Work (Existing Algorithm) on KC2

In order to evaluate the performance of our model against the existing software defect algorithms using in the benchmark paper, we use the KC2 dataset first, in each case we determined the mean square error (MSE) and error autocorrelation (R^2) obtained for comparison purpose, Table 2 presents the performance results of Support Vector Regression, Fuzzy Support Vector Regression, Decision Tree Regression, and the proposed approach on the complete KC2 dataset.

Table 2: Performance comparison against the benchmark work on KC2 software defect dataset

	SVR	FSVR	DTR	DPNN	Proposed
MSE	0.13	0.127	0.126	0.109	0.0439
R^2	0.193	0.228	0.234	0.297	0.9531

Table 2 presents the MSE and R^2 performance results on the KC2 datasets. The rows show the performance results of SVR, FSVR, DTR, and the proposed approach. The MSE and R^2 of SVR, FSVR, DTR, DPNN and the proposed approach on the KC2 dataset are (0.13, 0.193), (0.127, 0.228), (0.126, 0.234), (0.109, 0.297) and (0.0439, 0.9531) respectively. Table 2 is presented graphically for better understanding of the trend.

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

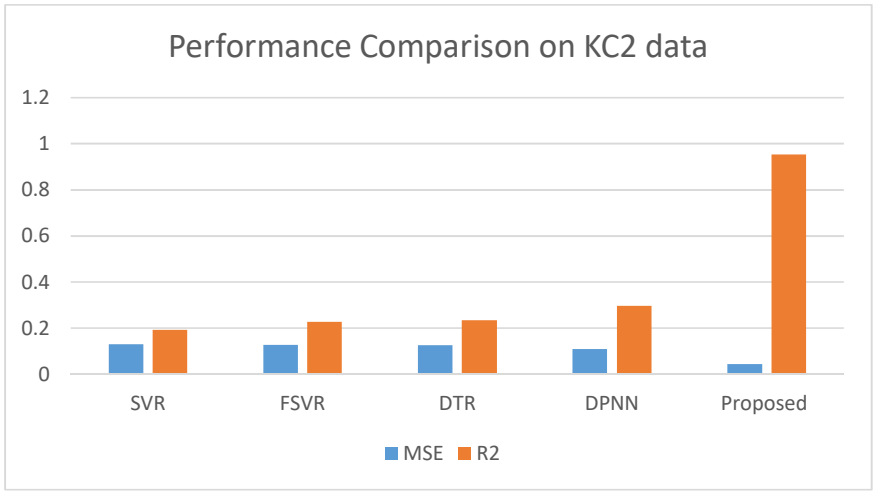


Figure 3. Performance comparison against the benchmark work on KC2 software defect dataset.

The MSE measures the quality of a prediction model and better values are non-negative and closer to zero. Thus, a smaller MSE represents better prediction model performance. Moreover, a smaller MSE means that the prediction model is closer to the optimal model. Hence, from Figure 3 above, the proposed approach achieved a very small MSE of 0.13 which implies a better performance as against the existing work of (Qiao et al., 2020) on the benchmark KC2 dataset. This is reasonably true since recurrent neural networks (RNN) are known to improve time series prediction errors compare to their counterpart ARIMA as shown by (Ashour & Abbas, 2018). (Ruiz, Rueda, Cuéllar, & Pegalajar, 2018)

Similarly, an R^2 value between 0 and 1 shows the extent to which the dependent variable is predictable, which represents how well the regression model fits the datasets. An R^2 value closer to 1 indicates that the model has achieved a good fit to this problem. A larger R^2 value represents a better model fit. Hence base on the simulation results recorded on table 2. The proposed model has a larger R^2 (0.9531) as against SVR (0.193), FSVR (0.228) and DTR (0.234) respectively, this can only signify a better model fitting by the proposed approach compare to the existing work.

Performance Comparison against Benchmark Work (Existing Algorithm) on MIS

Next, we will compare the performance of our proposed MLP-LSTM against the existing work for the MIS datasets, in each case we determined the mean square error (MSE) and error autocorrelation (R^2) obtained for comparison purpose, Table 3 presents the performance results of Support Vector Regression, Fuzzy Support Vector Regression, Decision Tree Regression, and the proposed approach on the complete MIS dataset.

Table 3. Performance comparison against the benchmark work on MIS software defect dataset

	SVR	FSVR	DTR	DPNN	Proposed
MSE	0.170	0.135	0.162	0.101	0.033
R^2	0.391	0.211	0.214	0.237	0.981

Table 3 presents the MSE and R^2 performance results on the MIS datasets. The rows show the performance results of SVR, FSVR, DTR, and the proposed approach. The MSE and R^2 of SVR, FSVR, DTR, DPNN and the proposed approach on the KC2 dataset are (0.170, 0.391), (0.135, 0.211), (0.162, 0.214), (0.101, 0.237) and (0.033, 0.981) respectively. Table 3 is presented graphically for better understanding of the trend.

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

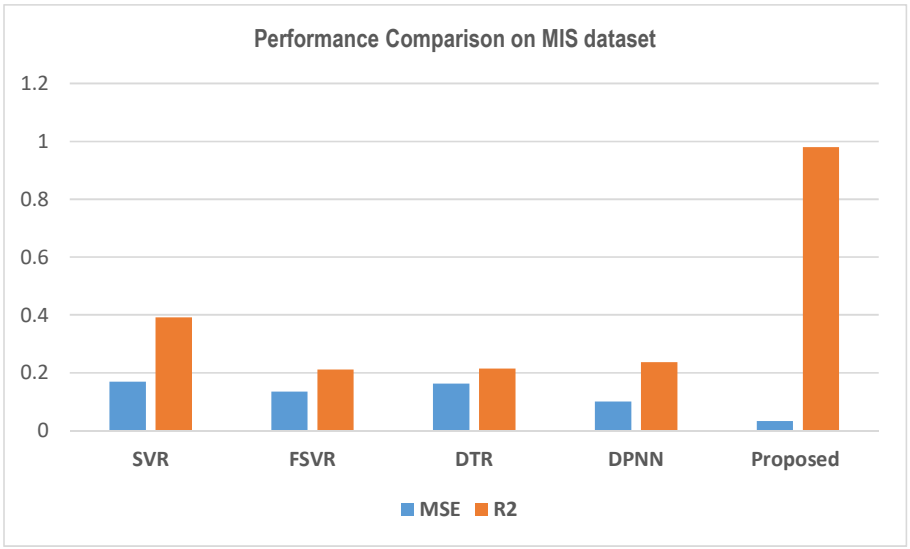


Figure 4. Performance comparison against the benchmark work on MIS software defect dataset.

From Figure 4 above, the proposed approach achieved a very small MSE of 0.13 which implies a better performance as against the existing work of (Qiao et al., 2020) on the benchmark MIS dataset. As stated earlier, this is reasonably true since recurrent neural networks (RNN) are known to improve time series prediction errors compare to their counterpart ARIMA as shown by (Ashour & Abbas, 2018). (Ruiz et al., 2018)

Similarly, as stated earlier, an R^2 value between 0 and 1 shows the extent to which the dependent variable is predictable, which represents how well the regression model fits the datasets. An R^2 value closer to 1 indicates that the model has achieved a good fit to this problem. A larger R^2 value represents a better model fit. Hence base on the simulation results recorded on table 3 and presented in Fig. 6. The proposed model has a larger R^2 (0.981) as against SVR (0.391), FSVR (0.211), DTR (0.214) and DTR (0.237) respectively, this implies a better model fitting by the proposed approach compare to the existing work.

Performance Evaluation of the Proposed MLP-LSTM On additional Datasets (JM1, KC1, CM1 and PC1)

In order to ensure the reliability of the proposed (MLP-LSTM) and extend the applicability of the proposed approach in the context of software defect prediction, we decided to test the proposed approach on four other different software defect datasets. A summary of all the performance achieved by the proposed MLP-LSTM is presented table 4 below for better understanding by the reader.

Table 4. Performance evaluation of the proposed model on all the five (5) software defect datasets

	CM1	JM1	KC1	KC2	PC1
MSE	4.7208E-5	4.1154E-7	5.5787E-7	0.0439	0.00048403
R^2	0.99886	0.99877	0.99994	0.9531	0.99639

Table 4 is further summarized and presented in Figure 4. For better understanding of the trend in the performance.

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

JOURNAL OF SCIENCE TECHNOLOGY AND EDUCATION 11(2), JUNE, 2023
ISSN: 2277-0011; Journal homepage: www.atbuftejoste.net

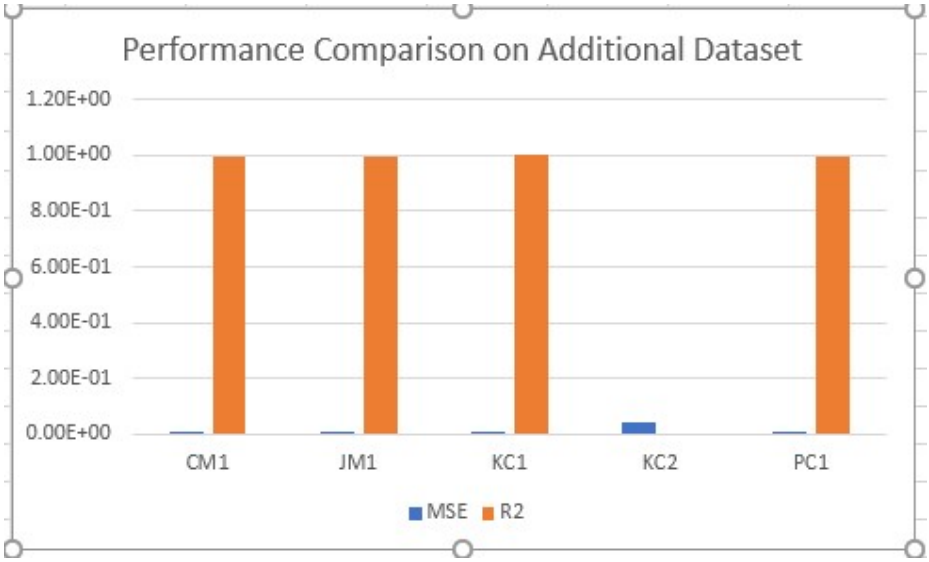


Figure 5: Performance Comparison on all the five datasets

From the simulations result depicted above in Table 4 and Figure 5 respectively, the proposed MPL-LSTM has performed well in all the five different datasets that were used for evaluation. The proposed MLP-LSTM achieved a very high autocorrelation value (R^2) of 0.99886, 0.99877, 0.99994, 0,9531 and 0.99639 for CM1, JM1, KC1, KC2 and PC1 respectively. An R^2 value closer to 1 indicates that the model has achieved a good fit to this problem. This means that, the proposed MLP-LSTM achieved a higher value (around 0.99 in all cases) which is very close to 1 on all the five evaluation datasets including: CM1, JM1, KC1, KC2 and PC1 respectively. This implies that the model has achieved a good fit to this problem. This is a significant improvement making the results of this research state of the art.

The results achieved in this research is reasonably true as expected since Recurrent neural networks (RNN) are known to improve time series prediction errors compare to their counterpart ARIMA as shown by (Ashour & Abbas, 2018). (Ruiz et al., 2018). This research further demonstrates the robustness of RNN in dealing with complex nonlinear data. In general, this research results demonstrate the suitability of the proposed model on different types of software defect datasets by achieving excellent results for MSE and R^2 on five different benchmark datasets.

CONCLUSION

Software failure happens when the software runs in an operational profile. Controlling failures in software require that one can predict problems early enough to take preventive action. The prediction of software failures is done by using the historical failures collected previously when they occur. Predicting defective code in the software development process is a crucial aspect of software analytics. Recently, a number of computational techniques were employed to solve this task. In particular, modern software is developed with more sizes and functions, and evaluating software reliability is a remarkably difficult task. These approaches have significantly facilitated the prediction of software defects; however, their performances (e.g., mean square error and squared correlation coefficient) requires significant improvements. The recently developed deep learning model, called deep neural network (NN) model, has appropriate prediction performance. This deep learning model do not only deepens the layer levels but can as well adapt to capture the training characteristics. A comprehensive, thorough study and feature excavation ultimately shows the model can have suitable prediction performance. Thus, this research work proposed a deep learning approach, specifically the (MLP-LSTM) in order to

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.

enhance the prediction performance of software defect based on deep learning algorithms using five benchmark datasets obtained from NASA project available at PROMISE repository portal. We evaluate the performance of the proposed model against the state-of-the-art approach using mean square error (MSE) and error correlation (R2). The key objective of this experiment is to employ a technique that improved upon the benchmark paper in terms of MSE and (R2). Base on the simulation results we observed and drive the following conclusions:

1. The MSE measures the quality of a prediction model and better values are non-negative and closer to zero. Therefore, a smaller MSE represents better prediction model performance. The MSE and R^2 of SVR, FSVR, DTR, DPNN and the proposed approach on the KC2 dataset are (0.13, 0.193), (0.127, 0.228), (0.126, 0.234), (0.109, 0.297) and (0.0439, 0.9531) respectively. Similarly, The MSE and R^2 of SVR, FSVR, DTR, DPNN and the proposed approach on the MIS dataset are (0.170, 0.391), (0.135, 0.211), (0.162, 0.214), (0.101, 0.237) and (0.033, 0.981). Hence, the proposed approach achieved a very small MSE which implies a better performance as against the existing work of (Qiao et al., 2020) on the benchmark KC2 and MIS dataset.
2. Similarly, an R^2 value closer to 1 indicates that the model has achieved a good fit to this problem. A larger R^2 value represents a better model fit. The proposed model has a larger R^2 (0.9531) as against SVR (0.193), FSVR (0.228) and DTR (0.234) respectively, this can only signify a better model fitting by the proposed approach compare to the existing work.
3. The proposed approach performed better in all cases of the software defect datasets CM1, JM1, KC1, KC2 and PC1 that was used for evaluating the experiment both in terms of MSE and R^2.
4. In general, this research results demonstrate the suitability of the proposed model on different types of software defect datasets by achieving low MSE values and Higher R^2 values.

LIMITATIONS AND FUTURE WORK

One of the major limitations of this study is that the study focuses on Forecasting Accuracy while ignoring the cost of computation that may be incurred by using two deep learners (MLP and LSTM). In our future work, should consider analysing the computational cost associated with these algorithms.

Acknowledgement

This study was supported by the Tertiary Education Trust Fund (TETFund) Institutional Based Research (IBR) Fund, through the directorate of Research and Innovation of Abubakar Tafawa Balewa University, Bauchi.

REFERENCE

Alhazmi, O. H., & Malaiya, Y. K. (2005). *Quantitative vulnerability assessment of systems software.* Paper presented at the Annual Reliability and Maintainability Symposium, 2005. Proceedings.

Aljahdali, S., & Sheta, A. F. (2011). *Predicting the reliability of software systems using fuzzy logic.* Paper presented at the 2011 Eighth International Conference on Information Technology: New Generations.

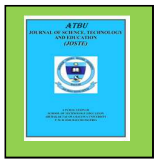
Ashour, M. A. H., & Abbas, R. A. (2018). *Improving Time Series' Forecast Errors by Using Recurrent Neural Networks.* Paper presented at the Proceedings of the 2018 7th International Conference on Software and Computer Applications.

Benaddy, M., El Habil, B., El Meslouhi, O., & Krit, S.-D. (2018). *Recurrent neural network for software failure prediction.* Paper presented at the Proceedings of the Fourth International Conference on Engineering & MIS 2018.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555.*

Cui, Z., Henrickson, K., Ke, R., & Wang, Y.

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.



(2019). Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems, 21*(11), 4883-4894.

Dam, H. K., Pham, T., Ng, S. W., Tran, T., Grundy, J., Ghose, A., . . . Kim, C.-J. (2018). A deep tree-based model for software defect prediction. *arXiv preprint arXiv:1802.00921*.

Fenton, N., Neil, M., & Marquez, D. (2008). Using Bayesian networks to predict software defects and reliability. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 222*(4), 701-712.

Ferenc, R., Bán, D., Grósz, T., & Gyimóthy, T. (2020). Deep learning in static, metric-based bug prediction. *Array*, 100021.

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2011). *The misuse of the NASA metrics data program data sets for automated software defect prediction.* Paper presented at the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011).

Gulcehre, C., Cho, K., Pascanu, R., & Bengio, Y. (2014). *Learned-norm pooling for deep feedforward and recurrent neural networks.* Paper presented at the Joint European Conference on Machine Learning and Knowledge Discovery in Databases.

Hata, H., Mizuno, O., & Kikuno, T. (2012). *Bug prediction based on fine-grained module histories.* Paper presented at the 2012 34th international conference on software engineering (ICSE).

Hu, Y.-L., & Chen, L. (2018). A nonlinear hybrid wind speed forecasting model using LSTM network, hysteretic ELM and Differential Evolution algorithm. *Energy conversion and management, 173*, 123-142.

Kalaivani, N., & Beena, R. (2018). Overview of software defect prediction using machine learning algorithms. *International Journal of Pure and Applied Mathematics, 118*(20), 3863-3873.

Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., & Ubayashi, N. (2012). A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering, 39*(6), 757-773.

Karim, S., Warnars, H. L. H. S., Gaol, F. L., Abdurachman, E., & Soewito, B. (2017). *Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset.* Paper presented at the 2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom).

Li, L., Lessmann, S., & Baesens, B. (2019). Evaluating software defect prediction performance: an updated benchmarking study. *arXiv preprint arXiv:1901.01726*.

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications, 144*, 113085.

Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2013). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

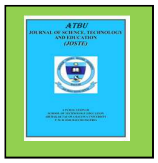
Qiao, L., Li, X., Umer, Q., & Guo, P. (2020). Deep learning based software defect prediction. *Neurocomputing, 385*, 100-110.

Ren, J., Qin, K., Ma, Y., & Luo, G. (2014). On software defect prediction using machine learning. *Journal of Applied Mathematics, 2014*.

Ruiz, L. G. B., Rueda, R., Cuéllar, M. P., & Pegalajar, M. (2018). Energy consumption forecasting based on Elman neural networks with evolutive optimization. *Expert Systems with Applications, 92*, 380-389.

Schröter, A., Zimmermann, T., & Zeller, A. (2006). *Predicting component failures at design time.* Paper presented at the Proceedings of the

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.



2006 ACM/IEEE international symposium on Empirical software engineering.

Sethi, T. (2016). *Improved approach for software defect prediction using artificial neural networks.* Paper presented at the 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO).

Soltani, R., & Jiang, H. (2016). Higher order recurrent neural networks. *arXiv preprint arXiv:1605.00064*.

Sun, Y., Xu, L., Li, Y., Guo, L., Ma, Z., & Wang, Y. (2018). *Utilizing Deep Architecture Networks of VAE in Software Fault Prediction.* Paper presented at the 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom).

Tamura, Y., Matsumoto, M., & Yamada, S. (2016). *Software reliability model selection based on deep learning.* Paper presented at the 2016 International Conference on Industrial Engineering, Management Science and Application (ICIMSA).

Wang, H., Lei, Z., Zhang, X., Zhou, B., & Peng, J. (2019). A review of deep learning for renewable energy forecasting. *Energy Conversion and Management, 198*, 111799.

Wang, J., & Zhang, C. (2018). Software reliability prediction using a deep learning model based on the RNN encoder–decoder. *Reliability Engineering & System Safety, 170*, 73-82.

Yan, M., Fang, Y., Lo, D., Xia, X., & Zhang, X. (2017). *File-level defect prediction: Unsupervised vs. supervised models.* Paper presented at the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).

Yang, Y., Zhou, Y., Liu, J., Zhao, Y., Lu, H., Xu, L., . . . Leung, H. (2016). *Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models.* Paper presented at the Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.

Corresponding author: Okoro Christian Chukwuka
okorochris63@gmail.com
Department of Mathematical Sciences, Abubakar Tafawa Balewa University, Bauchi.