International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

# Software Vulnerability Assessment and Classification Using Recurrent Neural Network and LSTM

**Ashok V. Markad[1], Dipak R. Patil[2], Bharat S. Borkar[3], Vilas S. Ubale[4], Sunildatta S. Kadlag[5], Manoj A. Wakchaure[6], Rohit N. Devikar[7]**

**Submitted**: 21/11/2023 **Revised**: 28/12/2023 **Accepted**: 09/01/2024

**Abstract:** Software defect detection is a valuable tool for enhancing the quality of technology and testing management. It allows for the quick identification of flaws in simulation models before the real testing phase begins. These prediction results assist technology designers in efficiently allocating their limited resources to areas that are more susceptible to shortcomings. This research presents a novel method for software bug prediction using deep learning techniques. A Recurrent Neural Network is used to classify source code, using various soft computing approaches. Data balancing for normalisation has included the use of many pre-processing and data filtering procedures. The generation of the Vector Space Model (VSM) has included the use of TF-IDF and related feature extraction approaches. The classification was performed using Recurrent Neural Networks (RNN) based on the training module for both the training and validation datasets. The proposed deep learning framework comprises many optimisation strategies, each with its own distinct advantages and constraints. We have assessed all methodologies and chosen the most superior one. To conduct the observations and test the suggested technique, a range of real-time and synthetic accessible datasets are assessed. The evaluation findings demonstrate that the proposed framework version surpasses both simple models of outstanding quality and complex deep learning classification models.

***Keywords:*** *LSTM, Bug prediction; ensembles; segmentation; classification; neural network; class imbalance learning; re-sampling methods; software defect prediction.*

## 1. Introduction:

The identification of weaknesses in source code or software has emerged as a prominent area of study. While previous research has shown the effectiveness of various detection techniques, models, and software vulnerability analysis tools in discovering source code vulnerabilities, improving the efficiency of these detection models and tools continues to be a significant problem for researchers. Every year, a large number of security vulnerabilities are discovered in virtual instruments. These vulnerabilities are then made public and added to a database that contains information on the weaknesses found in the code of these instruments. Threats may also manifest in indirect ways that may not be immediately apparent to code inspectors or programmers. It is essential to comprehend the intricacies of vulnerabilities that might directly cause system problems by analysing the raw data, which includes a vast amount of publicly accessible source code. This paper introduces an information-based approach to security technologies via the use of deep learning. Building upon the success of previous research in identifying vulnerabilities in source code and software, we use a theoretical framework to assess its potential in detecting these flaws. The first findings suggest that it is possible to define and identify assaults inside the given realm.

In order to address the gap between different domains, we may suggest that each feature in a programme be seen as a neural network in the field of computer vision. This is because fault detectors may just need to determine if a feature is unsecured and provide a comprehensive explanation of vulnerability situations. We want a discerning analysis of fault management programmes. It is advisable to consider each piece of code as a vulnerability detection unit, meaning that comments and code are utilised interchangeably in this research. Notably, there are significant exceptions to this diagnosis: (i) The majority of comments in a programme are likely to be certain, suggesting that only a small number of samples are vulnerable; and (ii) many comments are not considered as a cohesive unit that is semantically connected to one another.

A novel approach was developed to detect defects in huge volumes of random code by using classic programming techniques such as k-means cluster analysis and the generative adversarial model. The k-means cluster

[1,7] *Department of Information Technology, Amrutvahini COE, Sangamner, Maharashtra, India*

[2,6] *Department of Computer Engineering, Amrutvahini COE, Sangamner, Maharashtra, India* [4]*Department of Computer & Electronics Engineering, Amrutvahini COE, Sangamner, Maharashtra, India* [5]*Department of Electrical Engineering, Amrutvahini COE, Sangamner, Maharashtra, India* [1]*ashok.markad@avcoe.org,* [2]*dipak.patil@avcoe.org,* [3]*bharat.borkar@avcoe.org,* [4]*vilas.ubale@avcoe.org,* [5]*sunil.kadlag@avcoe.org,* [6]*manoj.wakchaure@avcoe.org,* [7]*rohit.devikar@avcoe.org*

transformation has been used to choose the most suitable code using an interactive analysis framework and software code rework generation [1]. The Tree-LSTM system, as described in the literature on object-oriented code analysis, utilises an LSTM network that operates as a tree structure. The model's validity is confirmed via the tasks of analysing conceptual relationships using coding pairs and identifying feelings [2]. Furthermore, this research study focuses on the kind of compilation that facilitates error detection for new developers while working with large source code. It also provides recommendations for addressing source code issues. The research use a previously established coding technique to identify vulnerabilities in particular code snippets [3]. An LSTM network would be used to forecast the level of motivation shown by the learners' instructor. A different research proposed a technique for detecting system software vulnerabilities using LSTM[4]. In order to determine the highest possible accuracy in classification, the number of neurons in the LSTM model was adjusted. The LSTM model achieves accurate outcomes in the detection of system software bugs [5].

Researchers often use traditional semi-supervised classifiers, recurrent neural networks (RNNs), long short-term memory (LSTM) models, or convolutional neural networks (CNNs) as classification methods for web application bug discovery and vulnerability classification. RNNs outperform typical language models, such as n-grams, although they have limitations in comprehending lengthy sequence data. LSTM, short for Long Short-Term Memory, is a variant of Recurrent Neural Networks (RNNs) that surpasses the constraints of traditional RNNs. The current research introduces a model that combines the process of consciousness with Long Short-Term Memory (LSTM) - Recurrent Neural Network (RNN). The LSTM-RNN network serves as a feature vector for the assessment and recognition of source code, relying on the anticipated maximum limit. The LSTM-AttM network outperforms the LSTM network, despite the fact that the former solely utilises the last hidden state outputs for estimate. When it comes to predicting activities, LSTM-RNN takes into account all the consequences of previously concealed states. The majority of the investigations have used different system software and classification methods to identify faults, functional programming, archive code, and basic error detection. Conversely, our proposed approach clearly identifies logic, grammar, and other system software flaws. Furthermore, instead of the erroneous location, the suggested model is used to forecast the accurate terms. In general, our proposed Language model differs from many previous models in its pursuit of distinct aims.

This study presents a new approach to active tracking in deep learning, which enables the automated learning of characteristics for anticipating vulnerabilities in runtime environments. When code components are located widely away in the source code, we use LSTM to capture enduring connections. For instance, when certain code tokens must occur together at the same time in a computer programme (e.g. in Java) or when they need to be used together based on the setting of an API (e.g. Lock() and active()), but they do not automatically come together, they are effectively managed. The acquired syntactic properties accurately represent the semantic functioning and hierarchical structure of the source code symbols. Our automated feature learning system eliminates the need for automated feature selection in traditional methods, hence reducing the significant amount of labour required. Ultimately, after subjecting the framework to extensive evaluation on several Java programmes for the Desktop edition, it becomes evident that our approach is exceptionally precise in elucidating code weaknesses.

## 2. Related Work :

Software defect prediction is used to identify potential defects throughout the software maintenance process with the aim of enhancing software dependability. The primary emphasis of conventional software defect prediction methodologies is on creating static code measurements, which are then used as input for machine learning classifiers to estimate the likelihood of code defects. Evaluating seven Apache open-source Java projects by using the area under the curve (AUC) and the F1 metric. The experimental results indicate that, on average, DP-ARNN surpasses the most advanced methods in terms of F1-measure and AUC by 14% and 7%, respectively [6].

Bug reports are a crucial and indispensable source of information for software development. When these reports are misclassified, bias is always introduced. Manual inspections may help reduce noise, but they impose a substantial burden on developers. In order to streamline the prediction process, Zhou et.al [7] proposed a multi-stage approach that included text mining and data mining methodologies.

An essential responsibility in the continuous development and upkeep of a programme is the resolution of software defects. Jin et.al [8] proposed a method to improve the accuracy of bug severity prediction. The bug reports for our classifier model comprise 'normal' severity bug reports, which constitute a significant proportion of all reported defects, together with the text and meta-fields.

Accurately determining the severity level of a bug report is an essential aspect in resolving bugs. To achieve equilibrium in the bug triaging process, it is necessary to categorise the severity of a bug report. To address these challenges, they propose a novel deep learning model called Bug Severity classification, which leverages Convolutional Neural Networks and Random Forest with

Boosting (BCR) for multiclass severity classification. This model directly learns the latent and highly representative characteristics. The bug report text is first subjected to preprocessing using natural language techniques, followed by the extraction of its features using n-grams [9].

The approach consists mostly of two phases: the two-stage ensemble (TSE) phase and the deep learning phase. Tong et. al [10] introduced a method where the DPs were originally derived from the traditional software metrics using SDAEs. They then tackled the problem of class-imbalance by using a unique ensemble learning technique called TSE.

## 3. Proposed System Design:

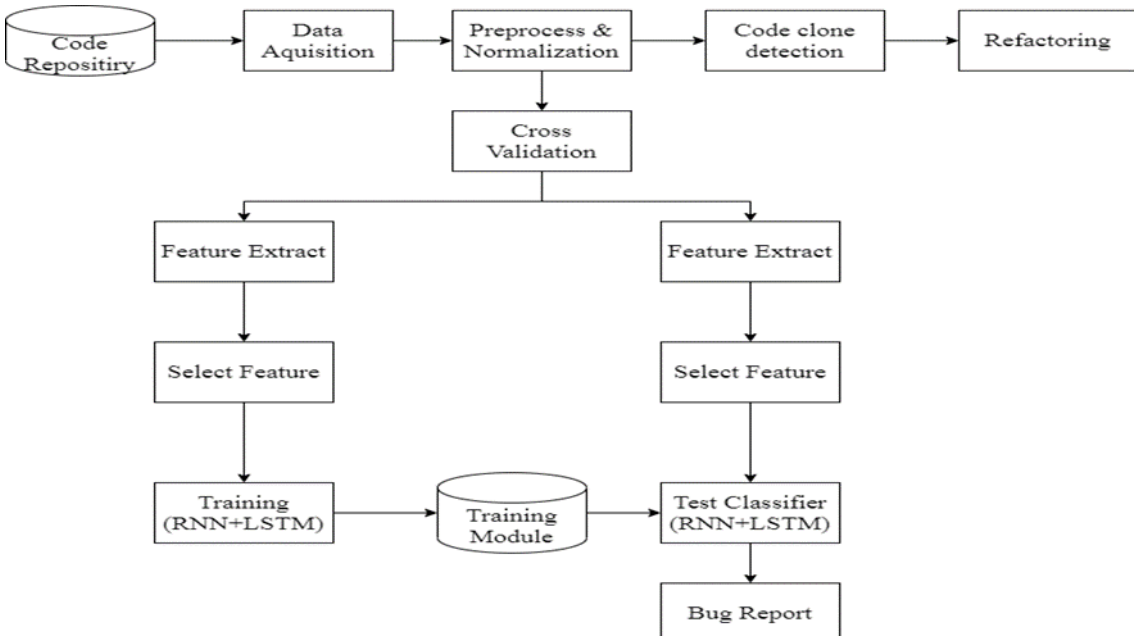


**Fig 1:** proposed system architecture design

A system overview of the execution process flow is shown in Figure 1, which can be seen above. This figure also outlines how the framework functions with various algorithms.

### 3.1 Feature Execution

A thorough compilation of the source code or modules is performed by the function, which also collects real statistics. The behaviour of the code is analysed using this method in order to locate any vulnerabilities. In order to construct the Vector Space Model (VSM) using the recovered characteristics during the course of the research, four different dynamic analysis procedures were used. These approaches were fault infusion, mutation suitable starting, dynamic taint assessment, and dynamic system check.

### 3.2. Data Pre-processing and normalization

At the beginning of the process of discovering a few clones, the source code is split, and the region of comparison is first agreed upon. In the phases that follow, there are three primary categories of objectives.

Remove the following portion of code: This step involves removing the source code that contains the boring comparison phase.

- Ascertain the units of origin and The remaining portion of the source code is separated into a series of distinct parts that are referred to as source units. This is accomplished by deleting all of the code that is not interesting.
- Determine the units and granularity of the correlation.
- portions of the source code have to be manually separated into smaller portions, with the division being determined by the evaluation approach that is used by a tool.

### 3.3 Extraction of heterogeneous features

The extraction process modifies the programme to its right form, while providing assistance for the actual comparison procedure. Depending on the device, it includes the following:

**Tokenization :** If an event involving token-based techniques occurs, each line of source code in the programme is divided into tokens according to the lexical rules of the programme design platforms. Utilise various tokens from source code lines or forms and then compare them using a token system framework. All whitespace and explanations among inscriptions are removed from the token groupings.

When it comes to parsing, syntactic techniques include describing the whole source code in order to build a parse tree or maybe a refined abstract syntax tree. After that, the source components that are going to be examined are shown as subordinate structures of the description tree or the tree, and correlation algorithms are used in order to find subordinate structures that are appropriate for verification as clones. Methodologies that are dependent on measurements could use a parse tree representation in order to detect clones, with the size of the sub trees being the determining factor.

The generation of dependencies from source code is accomplished via the use of semantic approaches in control and data flow analysis. The components of a Programme Dependence Graph are a visual representation of the reports and conditions of a system, while the edges of the graph display the control and information conditions. The subgraphs that include the source units that need to be matched are shown inside these PDGs. When looking for clones, one of the many ways that may be used is to search for isomorphic subgraphs. In order to compute information based on control stream data, many different ways make use of subgraphs.

### 3.4 Feature Selection

Different feature selection techniques have been used throughout the training of the module. The function compiles the complete source code or modules, capturing actual statistics. This technique analyses the behaviour of the code to find vulnerabilities. When considering a larger dataset, the importance of all variables becomes less significant. However, the more variables there are, the more challenging the analysis becomes. Consequently, it is sometimes more desirable to decrease the quantity of variables in a dataset and use crucial variables. To determine the value of a variable in a dataset, we may use a Function Selection strategy to decrease the parameter. Four dynamic analysis methodologies, including fault infusion, mutation appropriate beginning, dynamic taint evaluation, and dynamic system check, were used throughout the study. To create the Vector Space Model (VSM) from the collected characteristic.

### 3.5 Vulnerability Detection

The identification of vulnerabilities has been carried out by making use of the features that were gathered from the training dataset. developed on the basis of extracted characteristics such as TF-IDF, relational features, and certain bigram features, the vector space model has been developed. Recurrent neural networks, more especially the long short-term memory approach, were used in order to carry out the categorising process. The detection approach is also effective in avoiding attacks on web applications that are carried out using software as a service technologies. In addition to identifying and facilitating both internal and external attacks, the vulnerable code also makes it possible for unauthorised individuals to get access to the system. During the execution of code, the major objective of vulnerability detection is to automatically identify instances of attacks that include exception handling and buffer overflow. The strategy that is proposed in the code snippet provides increased detection accuracy in the sector that is specified.

## 4. Results and Discussion:

We have used RNN classification algorithms, which are well-suited for fault prediction even on unlabeled datasets, to verify the assessment of the suggested bug forecast method. The confusion matrix, which comprises the accuracy, recall, and F-score metrics, is used to evaluate software defect prediction ability (Table 1).

**Table 1:** confusion matrix evaluation

| **Actual** | **Predictive** | |
|---|---|---|
| **True** | TP (true positive) | FN (false negative) |
| **False** | FP (false positive) | TN (true negative) |

In order to assess the suggested system, we used machine learning classifiers such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Adaboost. In addition, we have implemented a deep learning framework. This framework utilises activation functions . The classification accuracy results, together with the confusion matrix, for all methods are shown in Table 2. The evaluation was performed using 20-fold cross-validation. The metrics used for algorithm comparison include Accuracy, Precision, Recall, and Micro-score. Based on the data, it can be inferred that the RNN (ReLU) achieves the greatest level of performance compared to the others.

### 4.1 Experiment using Artificial Neural Network:

Figure 2 displays the accuracy of the ANN classification method. Initially, it was trained using built-in functions from the Weka tool. Several cross-validation approaches have been used for classification, and different parameters have been adjusted for the artificial neural network (ANN) throughout the classification process. This technique may categorise each validation based on a

probability function, which is why this algorithm has a somewhat higher mistake rate compared to other supervised classification algorithms.

**Table 2:** Classification accuracy with confusion matrix for ANN

| ANN | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 85.20 | 84.20 | 85.60 |
| **Precision** | 83.60 | 82.30 | 84.99 |
| **Recall** | 87.50 | 85.40 | 77.72 |
| **Micro-Score** | 85.05 | 83.35 | 81.10 |

The Artificial Neural Network (ANN) model is straightforward to construct and especially valuable for the categorization of extensive datasets via the use of supervised machine learning techniques or Artificial Intelligence (AI). In addition to its simplicity, Artificial Neural Networks (ANN) are renowned for their ability to surpass even the most advanced categorization techniques. The suggested artificial neural network (ANN) forecasts the likelihood of an individual occurrence based on the present values.

Figure 2 depicts the assessment of the performance of Artificial Neural Network (ANN) classification using a 20-fold classification approach. The achieved accuracy for the provided input dataset is around 85.60%. We used a multinomial event model, where the samples indicate the frequency at which certain events have been created by a multinomial probability distribution. Based on this probability distribution, the system makes predictions for the final class.

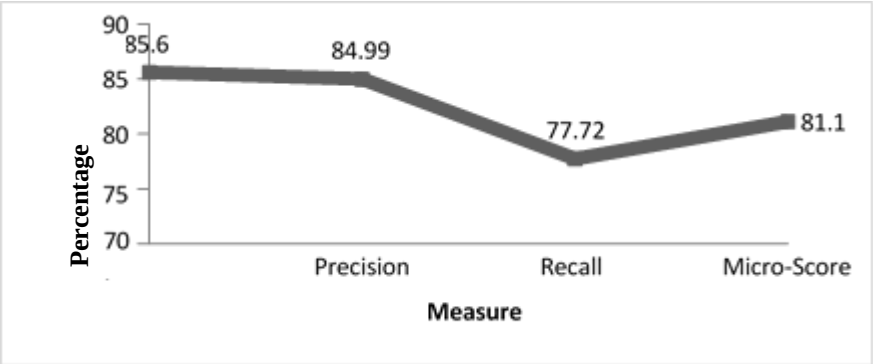


**Fig. 2:** An analysis of the discovery of bugs and vulnerabilities using artificial neural networks with twentyfold data cross validation

### 4.2 Analysis using SV model

According to the results of the classification study with different cross validations, which are shown in table 3 below, we have determined that the 20 fold cross-validation delivers the maximum classification accuracy for SVM, which is 95.2%.

**Table 3:** Classification accuracy with confusion matrix for SVM

| SVM | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 91.20 | 91.70 | 95.20 |
| **Precision** | 91.35 | 92.10 | 94.80 |
| **Recall** | 92.30 | 93.10 | 96.20 |
| **Micro-Score** | 91.35 | 92.20 | 94.75 |

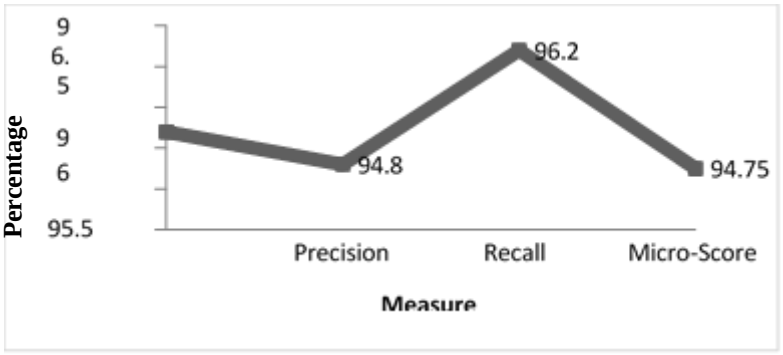


**Fig. 3:** Analysis of bug and vulnerability detection using SVM with 20-fold data cross validation

Figure 3 above illustrates the implementation of Support Vector Machine (SVM) classification for vulnerability detection using a 20-fold cross-validation approach. Generating a training set by labelling instances may be time-consuming and costly in many machine learning scenarios. Therefore, it is beneficial to explore methods for minimising the number of instances that need supervised classification. The Kernel-Based method has been used to minimise occurrences by enhancing performance. In this approach, we categorise everything as a point in n-dimensional spaces based only on the direction of a certain feature. We identify clones by locating the hyper-plane that effectively divides the two groups.

### 4.3 Experiment using Adaboost:

According to the results of the classification study with different cross validations, which are shown in table 4 below, we have determined that the 20 fold cross-validation delivers the maximum classification accuracy for Adaboost, which is 81.30%.

**Table 4:** Classification accuracy with confusion matrix for Adaboost

| Adaboost | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 70.60 | 78.50 | 81.30 |
| **Precision** | 72.30 | 73.50 | 74.50 |
| **Recall** | 69.90 | 68.50 | 70.30 |
| **Micro -Score** | 70.60 | 71.90 | 72.30 |

Adaboost is said to be adaptive since it modifies weak learners in the future in favour of situations that were incorrectly identified by previous classifiers. It is possible that, under some circumstances, it is less likely to experience the problem of overfitting than other learning methods. The whole model will converge to a strong learner as long as the performance of individual learners is considerably better than real guessing.

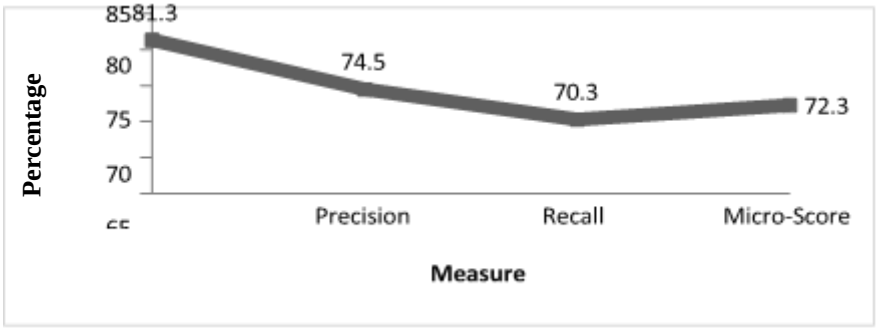


**Fig. 4:** Adaboost was used to do an analysis of bug and vulnerability detection, and 20-fold data cross validation was then performed.

Figure 4 provides a description of the Adaboost classification system for the identification of bogus accounts using 20-fold cross-validation. An example of a particular training method for boosted classifiers is the AdaBoost method.

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

The term "classifier" may also refer to a "boost classifier."

Each and every Ft is a weak learner that takes an object x as its input and then returns a result that denotes the class

of the object. In the case of the two-class problem, for instance, the sign of the weak learner output identifies the object class that is predicted, but the absolute value reflects the degree of confidence in that classification. In a similar vein, the Tth classifier is positive if the sample is a member of a positive class; otherwise, it is negative.

### 4.4 Experiment using Recurrent Neural Network (Sigmoid):

Within the scope of this experiment, we show the accuracy of the RNN (Sigmoid) classification algorithm by using a synthetic dataset. Similar tests have been conducted with a variety of cross validations, and the results are presented in table 5. Based on the findings of this investigation, we have determined that the implementation of RNN with Sigmoid function has the maximum classification accuracy of 96.10% when utilising 20 fold cross validation.

**Table 5:** Classification accuracy with confusion matrix for RNN (Sigmoid)

| RNN (Sigmoid) | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 95.60 | 95.90 | 96.10 |
| **Precision** | 95.80 | 96.10 | 97.00 |
| **Recall** | 95.80 | 96.00 | 96.30 |
| **Micro-Score** | 94.70 | 95.90 | 96.05 |

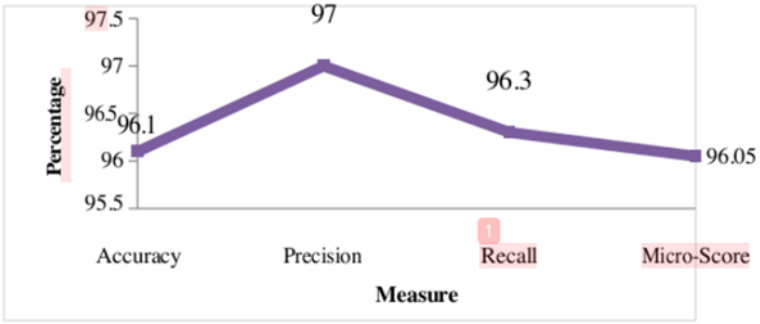


**Fig 5 :** Detection of accuracy using a Sigmoid Recurrent Neural Network (RNN) with 20-fold data cross validation

Based on the explanation provided in Figure 5, the 20-fold cross validation also achieves 96.10% using RNN with sigmoid function. This RNN functions accomplish about greater accuracy than the typical machine learning methods when it comes to module testing.

### 4.5 :Recurrent Neural Network, was used in the experiment with Tanh

The classification accuracy of RNN is shown in figure 6, and the results of comparable trials conducted with a variety of cross validations are presented in table 6. Based on the findings of this investigation, we have come to the conclusion that the maximum classification accuracy for RNN utilising Tanh is achieved via the use of 20 fold cross validation.

**Table 6:** Classification accuracy with confusion matrix for RNN (Tanh)

| RNN (Tanh) | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 96.90 | 97.50 | 97.25 |
| **Precision** | 97.00 | 97.40 | 97.60 |
| **Recall** | 97.30 | 97.50 | 97.30 |
| **Micro-Score** | 96.80 | 96.70 | 96.90 |

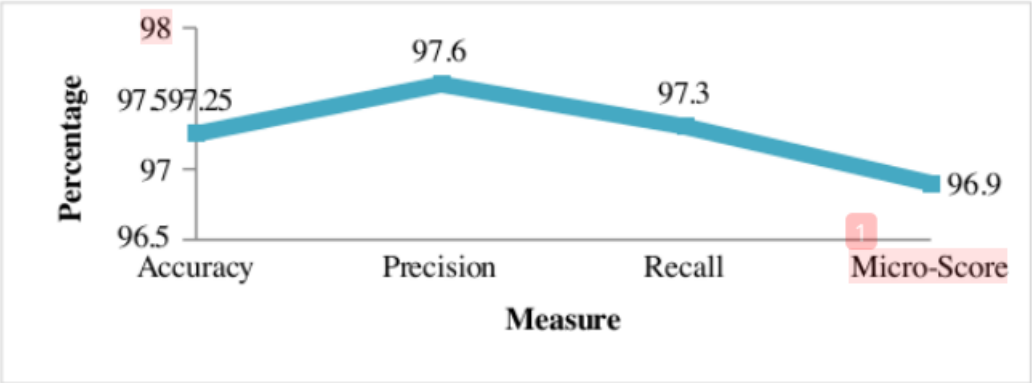


**Fig. 6:** Detection of accuracy using RNN (Tanh) with 20-fold data cross validation

### 4.5 Experiment using RNN (ReLU):

In this experiment, we investigate the accuracy of the ReLU classification algorithm by making use of a synthetic dataset. Previous experiments of a similar kind have been conducted using a variety of cross-validation methods, and the results are shown in table 7. Based on the findings of this investigation, we have come to the conclusion that the system offers the maximum accuracy of 97.5% for 20-fold cross validation classification accuracy for RNN..

**Table 7:** Classification accuracy with confusion matrix for RNN (ReLU)

| RNN (ReLU) | Fold 10 | Fold 15 | Fold 20 |
|---|---|---|---|
| **Accuracy** | 97.20 | 97.90 | 97.50 |
| **Precision** | 97.40 | 96.90 | 97.60 |
| **Recall** | 95.60 | 97.20 | 97.90 |
| **Micro-Score** | 96.20 | 95.80 | 97.20 |

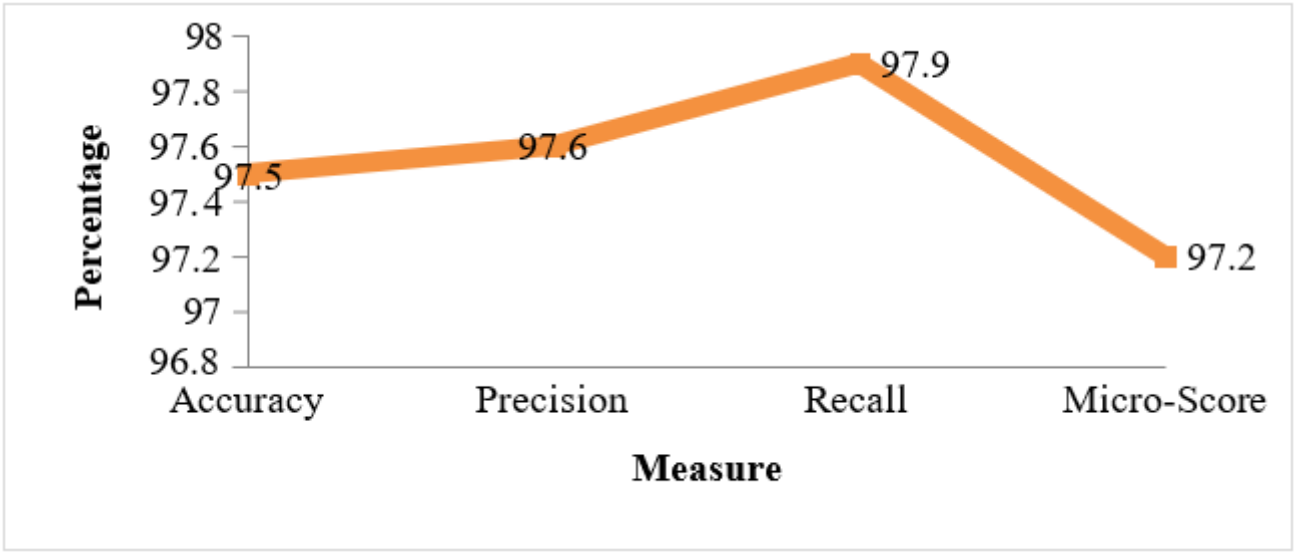


**Fig. 7:** Detection of accuracy using RNN (ReLU) with 20-fold data cross validation

A suggested deep learning classification method with a machine learning algorithm is described in the experiments that were shown before. The outcome is shown in this picture both with and without the assistance of cross-validation. The identification of code clone has been accomplished with the use of a minimum of three hidden layers. As a result of this experiment, we have come to the conclusion that the RNN with sigmoid detector offers superior detection accuracy in comparison to the other two activation functions and the random forest machine learning technique. A comparison of all of the outcomes of the tests described above can be seen in table 8.

**Table 8:** Classification accuracy with 20 folds cross-validation for all methods

| Method / Measure | ANN | SVM | Adaboost | RNN (Sigmoid) | RNN (Tanh) | RNN (ReLU) |
|---|---|---|---|---|---|---|
| **Accuracy** | 85.60 | 95.2 | 81.30 | 96.10 | 97.25 | 97.50 |
| **Precision** | 84.99 | 94.80 | 74.50 | 97.00 | 97.60 | 97.60 |
| **Recall** | 77.72 | 96.2 | 70.30 | 96.30 | 97.30 | 97.90 |
| **Micro-Score** | 81.10 | 94.75 | 72.30 | 96.05 | 96.90 | 97.20 |

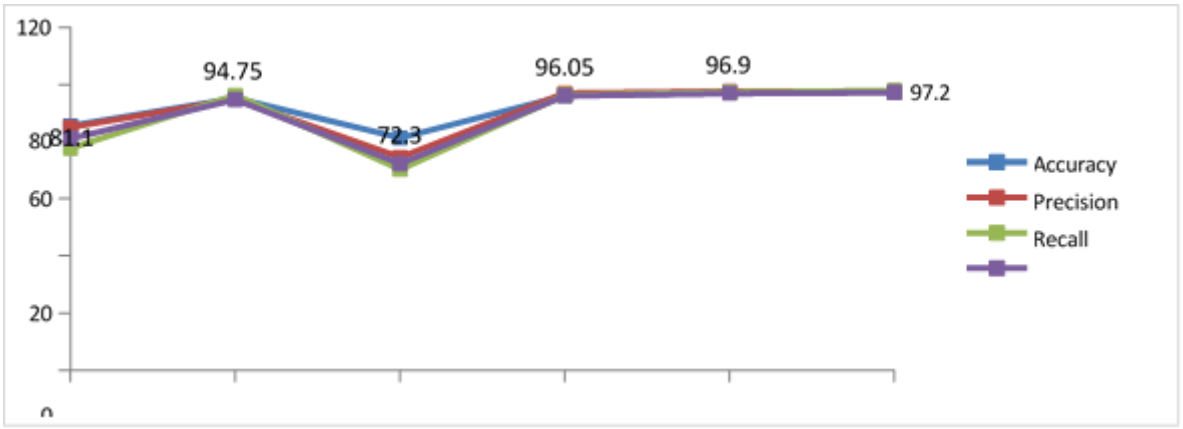


**Fig. 8:** Classification accuracy with 20-fold cross-validation for all methods

Another analysis demonstrated that our suggested solution had the highest prediction performance among the evaluated RNN algorithms, as shown by these assessment measures. Furthermore, certain datasets may not accurately capture all software problems and new defects due to limitations in data comprehensiveness and quantity. It is advisable to do further testing of the proposed solution inside real-world software applications. The three data splitting mechanisms are used in the form of 10-fold, 15-fold, and 20-fold cross-validation.

**Table 9:** An explanation of the source code that was taken from Android APK files as a dataset

| | |
|---|---|
| Total Size | 2500 |
| Training Samples | 2000 |
| Testing Samples | 500 |

The system provides a description of four assessments that compare the findings of this study with the findings of various current systems that have been computed on several datasets that are related to each other. This comparison between the suggested machine learning algorithms and some of the current ones is shown in Figure 9, which can be seen below.

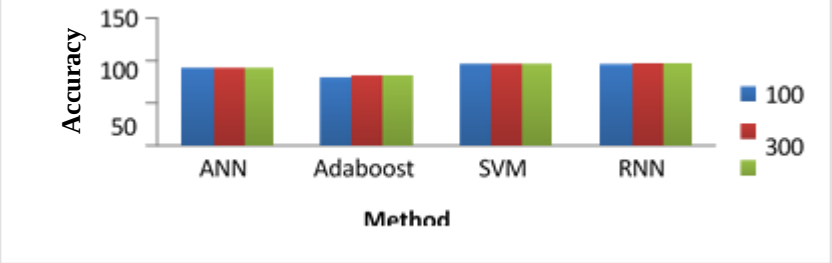


**Fig. 9:** A comparison of the proposed categorization with the current classification for the purpose of vulnerability detection

The Figure 9 displays the classification accuracy of the suggested methods for vulnerability detection, compared to two current machine learning techniques. This image illustrates that the suggested Recurrent Neural Network (RNN) offers superior accuracy in detecting compared to other machine learning techniques..

## 5. Conclusion and Future Work:

Detecting vulnerabilities in imbalanced source codes may be a laborious task. Vulnerable code might enable the

generation of software attacks on distant users. Occasionally, when the susceptible code is being executed, it may also give rise to internal attacks such as buffer overflow, session hijacking, and authentication bypass. Software literature annually identifies several issues with software. Vulnerabilities often do not manifest in concealed forms that can be readily identified by software testers. This system outlines the approach of identifying limitations by using deep learning techniques. This study presents the development of a Recurrent Neural Network (RNN) that incorporates Long Short-Term Memory (LSTM) for the purpose of creating code vulnerability detection and bug triage systems across several platforms. Many technologies lack the capability to assist a web-based application in identifying code vulnerabilities. The suggested system is capable of detecting vulnerabilities by performing feature extraction on various datasets. Recurrent Neural Networks (RNN) provide superior outcomes compared to conventional machine learning classifiers. In the future, developers will want the ability to identify code triage for runtime mobile-based application programmes, since the current tools lack support for such programmes. Code clone management is another need in software engineering. Superior design quality may be attained by the use of glitches. Open-source code replication in software development.

## References:


[1] Terada, K.; Watanobe, Y., "Automatic Generation of Fill-in-the-Blank Programming Problems", In Proceedings of the 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 1–4 October 2019; pp. 187–193.

[2] Tai, K.S.; Socher, R.; Manning, C.D., "Improved semantic representations from tree-structured long short-term memory networks", In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; pp. 1556–1566.

[3] Pedroni, M.; Meyer, B., "Compiler error messages: What can help novices?", In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, Portland, OR, USA, 12–15 March 2008; pp. 168–172.

[4] Saito, T.; Watanobe, Y., "Learning Path Recommendation System for Programming Education based on Neural Networks", Int. J. Distance Educ. Technol. (Ijdet) 2019, 18, 36–64.

[5] Teshima, Y.; Watanobe, Y., "Bug detection based on LSTM networks and solution codes", In Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 7–10 October 2018; pp. 3541–3546.

[6] Fan, G.; Diao, X.; Yu, H.; Yang, K.; Chen, L., "Software Defect Prediction via Attention-Based Recurrent Neural Network", Sci. Program. 2019, 2019, 6230953.

[7] Zhou, Y.; Tong, Y.; Gu, R.; Gall, H., "Combining text mining and data mining for bug report classification", J. Softw. Evol. Process 2016, 28, 150–176.

[8] Jin, K.; Dashbalbar, A.; Yang, G.; Lee, J.-W.; Lee, B., "Bug severity prediction by classifying normal bugs with text and meta-field information", Adv. Sci. Technol. Lett. 2016, 129, 19–24.

[9] Goseva-Popstojanova, K.; Tyo, J., "Identification of security related bug reports via text mining using supervised and unsupervised classification", In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, Lisbon, Portugal, 16–20 July 2018; pp. 344–355.

[10] Tong, H.; Liu, B.; Wang, S., "Software defect prediction using stacked denoising auto encoders and two-stage ensemble learning", Inf. Softw. Technol. 2018, 96, 94–111.

[11] Singh, H., Ahamad, S., Naidu, G.T., Arangi, V., Koujalagi, A., Dhabliya, D. Application of Machine Learning in the Classification of Data over Social Media Platform (2022) PDGC 2022 - 2022 7th International Conference on Parallel, Distributed and Grid Computing, pp. 669-674.

[12] Veeraiah, D., Mohanty, R., Kundu, S., Dhabliya, D., Tiwari, M., Jamal, S.S., Halifa, A. Detection of Malicious Cloud Bandwidth Consumption in Cloud Computing Using Machine Learning Techniques (2022) Computational Intelligence and Neuroscience, 2022, art. no. 4003403, .