International Journal on Advances in Engineering Technology and Science
Volume: 4, Issue: 1, 2023, e-ISSN: 2455-3131

Forward Pass RNN and Hyperbolic Mapping used in Software Bug Prediction

Pooja Singh[1], Rupali Chaure[2], Ritu Shrivastava[3]

[1]Research Scholar, [2]Assistant Professor, [3]Head and Professor
Department of CSE, SIRT, Bhopal, India

Abstract— **In academics and industry, software bug prediction (SBP) is essential for assessing worker dependability. Early fault discovery enhances software adaption, efficacy, user happiness, and resource efficiency. Early in the software development lifecycle, a variety of measurements and techniques are used. The goal is to enhance the accuracy, recall, and precision of software problem detection in retrieving relevant flaws. By merging FPRNN with Hyperbolic Mapping, the FPRNN-HM approach improves software defect prediction by speeding up convergence and enhancing searching power, ultimately identifying ideal attributes. The FPRNN-HM model achieves high accuracy of 98.45% for big datasets, prevents overfitting, and offers high computation, making it an affordable tool for software development bug prediction.**

***Keywords*— SBP, FPRNN-HM, Accuracy, Precision.**

I. INTRODUCTİON

The impact of software programmes is growing every day. The evaluation of reliability in labour is becoming more and more significant in both industry and academics. Improving software quality with limited testing resources as software testing duration and total cost continue to rise is an essential test for any researcher or software professional. Sorting software modules into problematic and non-faulty categories is the main objective of software bug prediction (SBP) approaches. To enhance software quality, the engineer will thereafter propose options for testing different software modules and practical test resources [1].

The frequency of software defects significantly affects the programme's performance, reliability, and cost of operation. Even with proper usage, it takes a lot of work to produce software free of bugs since hidden flaws are often present [2].

Developing a model for software bug prediction that can detect malfunctioning modules early on is a significant challenge in software engineering [3]. One essential phase in the software development process is the prediction of defects [4]. This is because detecting troublesome modules prior to programme deployment improves user satisfaction and overall software effectiveness [5]. Furthermore, early software issue prediction improves resource efficiency and software adaptation to different environments. Several software metrics, including class level, method level, fle level, and process level, are used in the early stages of the software development life cycle to find software flaws without actually testing the programme [6, 7]. Numerous methods, such as statistical analysis, machine learning, expert systems, etc., may be used to find software flaws.

In the present study, we propose the FPRNN-HM (Forward Pass RNN with Hyperbolic Mapping) method to improve software fault prediction. Selecting the most effective characteristics that might reveal the underlying structures of the defect data is crucial for developing effective defect prediction models. The main contribution of the proposed model is listed below:

- The FPRNN (Forward Pass RNN) with method and HM (Hyperbolic Mapping) are utilised to find software flaws.
- The suggested FPRNN (Forward Pass RNN) combines HM (Hyperbolic Mapping). The purpose of this technique is to accelerate convergence and improve searching capability.
- The recommended approach makes use of the HM (Hyperbolic Mapping) to choose the best characteristics.
- To find the software problem, HM (Hyperbolic Mapping) is suggested here.

In particular, SBP is covered in great detail in this work and is divided into the following subsections: In Section 2, the history of SBP is shown. The issue identification of prediction analysis is shown in Section 3. Section 4 explains the study aims, and Section 5 provides an overview of the SBP approach using data from specific dataset. Results and current developments in prediction analysis are shown in Section 6.

II. BACKGROUND

Several academics used artificial intelligence approaches to uncover software bugs. A selection of these works are mentioned below; More than 4000 defect reports were collected from three open-source database systems and

mechanically classified using the Orthogonal Defect Classification (ODC) method, according to Lopes et al.'s analysis [8]. Undersampling was used to get around uneven datasets. The findings of the experiment show that categorising certain ODC properties automatically using only reports is difficult. Similar to this, semi-supervised learning-based automatic ODC defect-type classification was carried out by Thung et al. [9]. In this instance, 500 problem reports gathered from three software systems were classified. The utilisation of huge datasets will have an impact on classification accuracy. Three factors were used by Tan et al. [10] to construct a bug classification system: impact, dimensions root cause, and affected component. The machine learning methods were used for the classification procedure. By using machine learning methods, 109,014 bugs are automatically found. A machine-learning approach was presented by Li et al. [11] to examine bug characteristics in open-source software. They proposed the classification of a problem based on concurrent memory and semantic flaws, much as Tan et al. The programming studies and cypher excellence of open-source projects were examined by Ray et al. in [12]. They introduced machine learning classifiers to accomplish this goal. By using abstract syntax trees (ASTs) and tree-based coding, Ni et al. [13] predicted root cause categories (TBCNN). There are 21 subcategories and six primary types of origin reasons.

Goseva et al. [14] used supervised and unsupervised learning methods to analyse mistakes based on security and non-security. High-impact mistakes were predicted by Wu et al. [15] using machine learning approaches for active learning. A machine learning approach and Fecher selection method for predicting Mandelbucks and Borbucks were described by Xia et al. [16]. Subsequently, a system for cross-project domain adaptation with the same purpose was created by Du et al. [17]. Additionally, [18] provides a straightforward explanation of error detection and a positive impression of articles on error classification and prioritisation.

A machine learning (ML) algorithm-based technique for software bug prediction was reported by Hammouri et al. [19] in 2018. Using three closely watched machine learning approaches, potential software problems were predicted based on past data. The assessment method demonstrated the proper and effective application of ML algorithms. Empirical results showed that the ML methodology is more effective for the estimate procedure than other strategies like linear AR and POWM models. In order to create, develop, and evaluate bug forecasting models in real-world continuous software evolution situations, Wang et al. [20] examined software bug prediction. ConBuild uses the diferential qualities of bug prediction data to rethink how training data is selected for models. ConEA uses fle-bug probability growth to redefine effort-aware evaluation in continuous software development. The utility of techniques is shown by analyses of 120 regularly published versions of six large-scale open-source software systems.

Artificial Immune Networks (AIN) and machine learning classifiers based on software bug detection were examined by Khan et al. [21]. The hyperparameters were chosen best to boost the bug prediction process's dependability. A paradigm for an object-oriented software bug prediction system (SBPS) was examined by Gupta and Saxena [22]. A few open-source projects with similar issue datasets were obtained for this study via the Promise Software Engineering Repository. The Logistic Regression Classifer has the highest accuracy of all the classifiers.

Software bug fault identification methods that use the collective sorting approach were examined by Moustafa et al. in [23]. The techniques were tested on datasets of different sizes and used to apply different software measurement groups as sorting algorithm characteristics. The results showed that update measurements outperformed both an approach that combines equal amounts of data and static code measures. Qu and Yin [24] assessed network embedding approaches in bug identification and used node2defect, a flaw detection framework that concatenates integrated vectors using traditional software engineering metrics, to create and improve on it. The trials employed 13 open-source Java systems, two effort-aware models, and seven connection embedding approaches.

A new review [25] provides a thorough explanation of the use of deep learning methods in software development research, including the forecasting of flaws and vulnerabilities and the localization of errors. Huang et al. [26] manually classified approximately 5,400 phrases from published papers into seven categories, including "Information Delivery" and "Problem Discovery." They later created a deep neural network to predict these goals. Localising software bugs was created by Mahajan and Chaudhary [27]. A hybrid optimization-based CNN was created to accomplish this goal. For feature selection, they presented a hybridised cuckoo search-based sea lion optimisation method. When compared to other approaches, the procedure produced excellent results. Deep reinforcement learning technique-based bug identification in video games was first shown by Rani et al. [28]. A graph CNN-based software version-to-version bug prediction system was created by Wang et al. [29]. The analysis of deep learning algorithms based on bug prediction was done by Choetkiertikul et al. [30]. Software bug detection based on feature transformation was developed by Cynthia et al. [31]. In this case, feature selection-based prediction was the primary emphasis. Furthermore, a deep learning algorithm-based bug prediction was created by Giray et al. [32]. Here, they examined the performance of several machine learning and deep learning methods.

Many researchers focused on deep learning approaches and prediction based on machine learning algorithms while examining the literature review. The majority of the researchers in this processed all the information without concentrating on the best aspects. Both the complexity and time required for computing will grow as a result. This research proposes feature selection-based software bug prediction as a workaround for the problems.

III. PROBLEM IDENTİFİCATİON

The problems identified by previous research are as follows [1, 4]:

- It is not always possible to identify relevant software flaws.
- A software bug's recovery is not entirely recognized.
- The poor accuracy of the unnamed software issue may lead to its detection.

IV. RESEARCH OBJECTİVES

The aims of the suggested work are as follows:

- To increase accuracy in order to perfectly retrieve pertinent software flaws.
- To increase recall for software faults that are absolutely relevant throughout the retrieval process.
- To increase the precision of software problem detection.

V. METHODOLOGY

The algorithm of proposed model is as follows:

I = Number of input layers

H = Number of hidden layers

O = Number of output layers

S = Number of data set instances

Step 1: for i = 1 to H

Step 2: for j = 1 to S

calculating the forward for the forward hidden layers with activation function

$$h_t^f = \tanh\left(W_h^f h_{t-1}^f + W_x^f x_t + b_h^f\right) \quad (1)$$

end for

Step 3: for j=S to 1

calculating the backward pass for the backward hidden layer's activation function

$$h_t^b = \tanh\left(W_h^b h_{t-1}^b + W_x^b x_t + b_h^b\right) \quad (2)$$

end for

end for

Step 4: for i =1 to O

calculating the forward pass for the output layer using the previous stored activation function

$$P\left(y_t \left| \{x_i\}_{i\neq t}\right.\right) = \sigma\left(W_y^f h_t^f + W_y^b h_t^b + b_y\right) \quad (3)$$

Wy is the weight matrix connecting the hidden layer to output layer, Wh is the weight matrix that connects hidden to hidden layer, and Wx is the weight matrix that connects input layer to hidden layer. by is the output layer bias vectors, and bh is the hidden layer bias vectors. For the final nonlinearity r, and use tanh as an activation function for classification. According to this form, the RNN will evaluate the output yt according to the information propagated through the hidden layer regardless of whether it depends directly or indirectly on the values

$$\{x_i\}_{i=1}^t = \{x_1, x_2, ..., x_t\} \quad (4)$$

VI. EXPERİMENT AND RESULT

This section outlines the precise procedures of the experiment after outlining various presumptions and constraints. The following are the presumptions made in this work:

(1) The training and testing data come from a single dataset, and our main emphasis is on software defect prediction inside a project. When using the ant dataset for experimentation, for instance, the training set is chosen and the test set is created from the remaining portion of the dataset.

(2) The trained model favours the non-faulty classes during the trials because of a limited number of defective classes. Thus, before training the model, class imbalance is applied to the whole dataset.

(3) A tenfold cross-validation is used in order to more accurately measure the algorithm's performance.

The software defect prediction system presented in this study can be validated under the aforementioned assumptions. The particular protocol for the experiment is as follows:

Step 1: The software's class dependency is extracted using the code analysis tool, and a CSV file is subsequently created.

Step 2: The PROMISE dataset is used to extract the labelled nodes and feature metrics for each node.

Step 3: To address data class imbalance, the FPRNN-HM technique is used.

The following observations are made using Python 3.11.1 and the Jupyter Lab toolbox on Anaconda Navigator. The following formulas are used to determine the precision, recall, F1-Score, and accuracy parameters of the suggested FPRNN-HM process using the (PROMISE Dataset) JS1.csv:

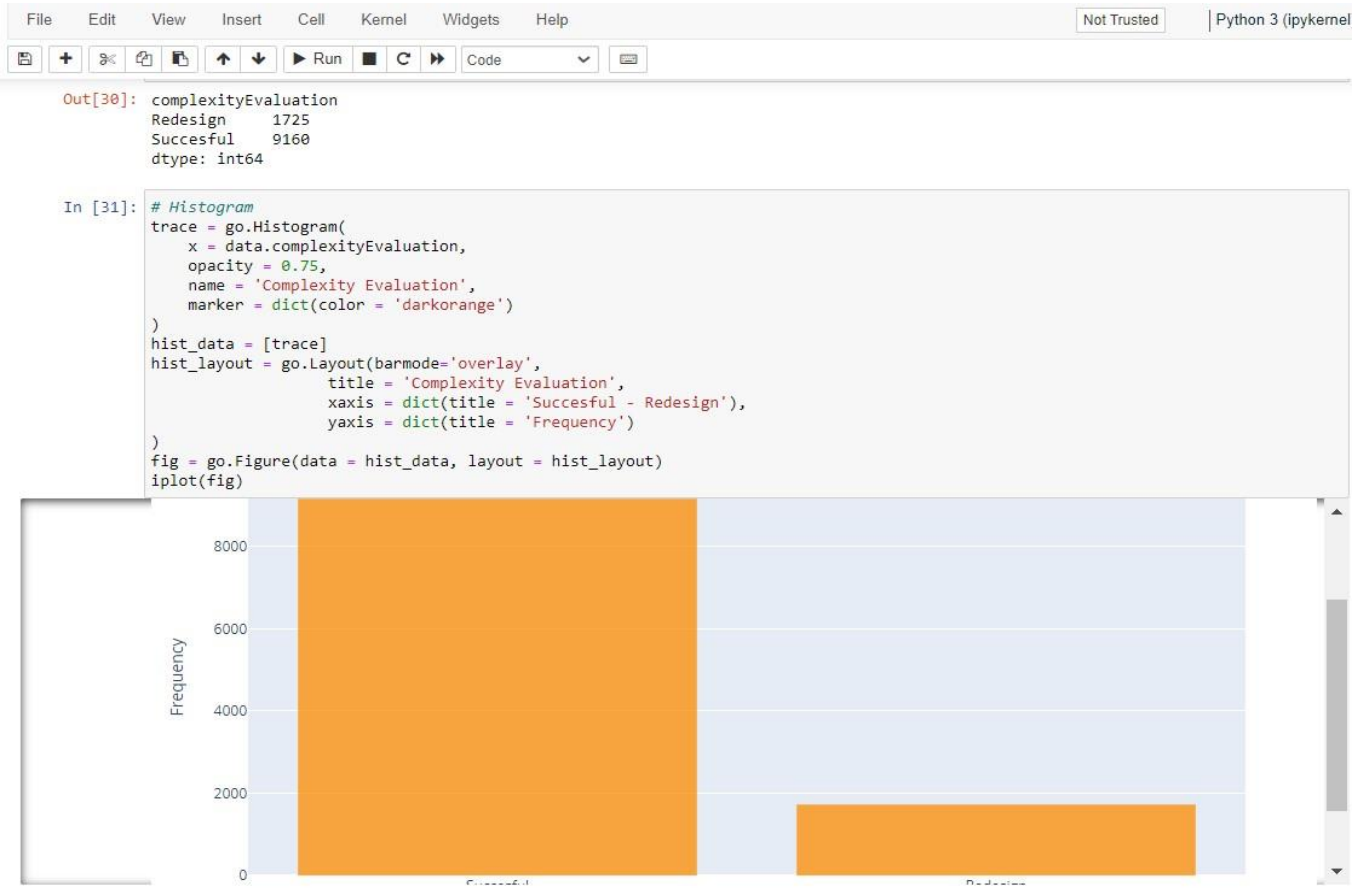

Figure 1. Evaluation of Bug Frequency in Software for FPRNN-HM (Proposed Prediction Model)

Table 1. Estimation of Confusion Matrix among different models and FPRNN-HM (Proposed Prediction Model)

Models	Prediction	Module has bugs	
		No	Yes
Random Forest	Classifier predicts no bugs	1652	80
	Classifier predicts some bugs	341	103
Naïve Bayes	Classifier predicts no bugs	1663	69
	Classifier predicts some bugs	364	80
Logistic Regression	Classifier predicts no bugs	1701	31
	Classifier predicts some bugs	415	29
Decision Tree	Classifier predicts no bugs	1453	279
	Classifier predicts some bugs	268	176
ANN	Classifier predicts no bugs	1738	36
	Classifier predicts some bugs	339	63
FPRNN-HM (Proposed)	Classifier predicts no bugs	1841	13
	Classifier predicts some bugs	19	304

Table 2. Estimation of Precision, Recall, F1-Score and Accuracy among different models an FPRNN-HM (Proposed Prediction Model)

Models	Precision	Recall	F1-Score	Accuracy
Random Forest	0.95	0.83	0.9	80.65 %
Naïve Bayes	0.96	0.82	0.88	80.10 %
Logistic Regression	0.98	0.8	0.88	79.5 %
Decision Tree	0.83	0.84	0.83	74.86 %
ANN	0.97	0.84	0.9	82.77 %
FPRNN-HM (Proposed)	0.99	0.98	0.99	98.45 %

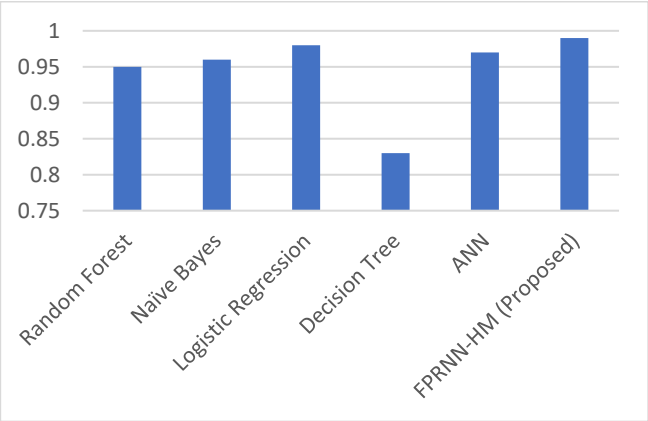

Figure 2. Graphical Analysis of Precision among different models and FPRNN-HM (Proposed Prediction Model)

The above graph show that the proposed model gives better precision for bug prediction as compare than other models. The precision of FPRNN-HM is improved by 0.01 as compare than Logistic Regression prediction model.

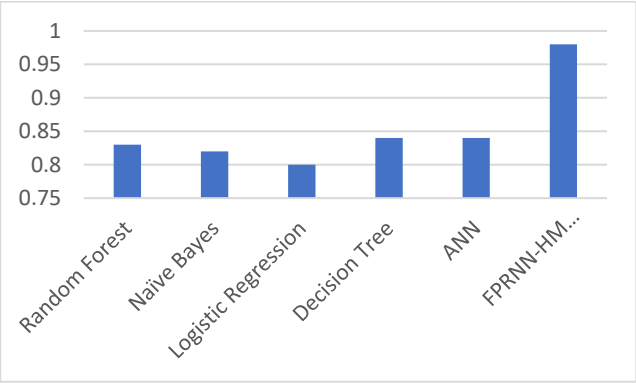

Figure 3. Graphical Analysis of Recall among different models and FPRNN-HM (Proposed Prediction Model)

The above graph show that the proposed model gives better recall for bug prediction as compare than other models. The recall of FPRNN-HM is improve by 0.14 as compare than Decision Tree and ANN prediction model.

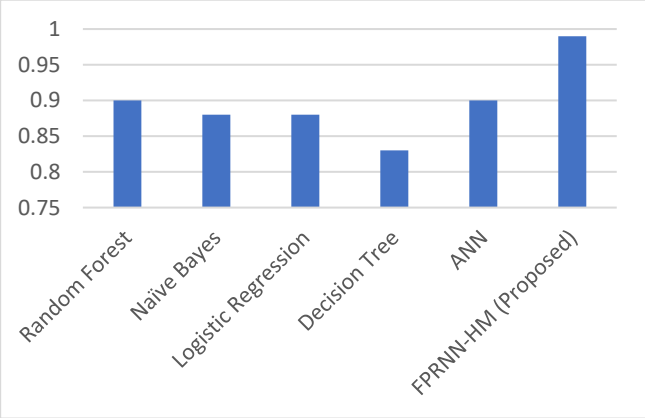

Figure 4. Graphical Analysis of F1-Score among different models and FPRNN-HM (Proposed Prediction Model)

The above graph show that the proposed model gives better F1-Score for bug prediction as compare than other models. The F1-Score of FPRNN-HM is improve by 0.09 as compare than Random Forest and ANN prediction model.

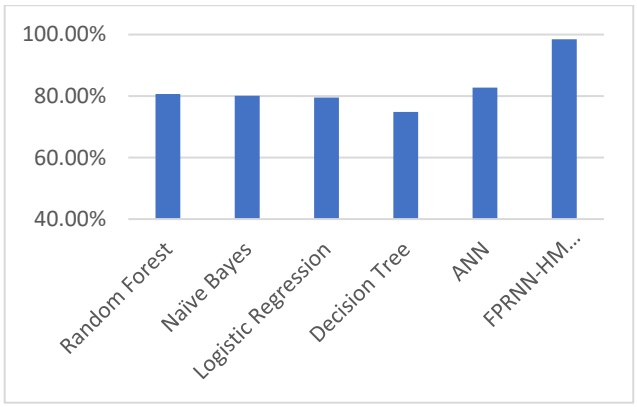

Figure 5. Graphical Analysis of Accuracy among different models and FPRNN-HM (Proposed Prediction Model)

The above graph show that the proposed model gives better Accuracy for bug prediction as compare than other models. The Accuracy of FPRNN-HM is improved by 15.68 % as compare than ANN prediction model.

VII. Conclusions

Since bug prediction reduces production costs, maintenance costs, and dependability, it is essential in the early stages of software development. To create a successful programme, we have used the FPRNN-HM model in our suggested work. Our findings demonstrates that feature selection and cross-validation were not given enough priority in previous research. When compared to other methods, the suggested technique produces high accuracy of 98.45% for huge datasets, indicating that it is the best. The optimal combination of the five created algorithms is simple to use, predicts more accurately, prevents overfitting, offers high computation, can be used to both regression and classification tasks, and performs well with huge datasets. Even now, research is being done to understand more about this approach to bug prediction in conjunction with a machine learning model.

The following are the work's conclusions:

1. The suggested model outperforms ANN in terms of prediction accuracy. There is a 15.68% increase in accuracy.

2. The suggested model outperforms logistic regression in terms of prediction accuracy. There is a 1% increase in accuracy.

3. The suggested model outperforms Decision Trees and ANNs in terms of prediction recall. Recall gains 14% better.

4. The suggested model outperforms ANN and Random Forest in terms of prediction F1-Score. By 9%, the F1-Score becomes better.

Our suggested technique is very beneficial for future development and helps to increase the accuracy of bug prediction. In order to verify the accuracy estimate in future improvements, the accuracy must be validated using other datasets and other AI algorithms. Because so much data was collected to estimate the performance of the train data, the processing time of the suggested model is limited. The same methods will be used in the future to estimate the system's efficacy using real-time data.

REFERENCES

[1] R. Siva, Kaliraj S, B. Hariharan, N. Premkumar (2023). Automatic Software Bug Prediction Using Adaptive Artifcial Jelly Optimization with Long Short-Term Memory. Wireless Personal Communications (pp. 1975-1998). Springer.

[2] Abozeed, S.M., ElNainay, M.Y., Fouad, S.A. & Abougabal, M.S. (2020). Software bug prediction employing feature selection and deep learning. In 2019 International Conference on Advances in the Emerging Computing Technologies (AECT) (pp. 1–6). IEEE.

[3] Panda, M. & Azar, A.T. (2021). Hybrid multi-objective grey wolf search optimizer and machine learning approach for software bug prediction. In Handbook of research on modeling, analysis, and control of complex systems (pp. 314–337). IGI Global.

[4] Kumar, R., & Gupta, D. L. (2016). Software bug prediction system using neural network. European Journal of Advances in Engineering and Technology, 3(7), 78–84.

[5] Chaubey, P.K., & Arora, T.K. (2020). Software bug prediction and classifcation by global pooling of diferent activation of convolution layers. Materials Today: Proceedings.

[6] Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., & Gyimóthy, T. (2020). An automatically created novel bug dataset and its validation in bug prediction. Journal of Systems and Software, 169, 110691.

[7] Aggarwal, A., Dhindsa, K.S., & Suri, P.K. (2021). Enhancing software quality assurance by using knowledge discovery and bug prediction techniques. In Soft computing for intelligent systems (pp. 97–118). Springer, Singapore.

[8] Thung, F., Le, X.B.D., Lo, D. (2015). Active semi-supervised defect categorization. In: 23rd Int. conference on program comprehension, pp 60–70.

[9] Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., & Zhai, C. (2014). Bug characteristics in open source software. Empirical Software Engineering, 19(6), 1665–1705.

[10] Zhang, N., Ying, S., Ding, W., Zhu, K., & Zhu, D. (2021). WGNCS: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation. Information Sciences, 570, 545–576.

[11] Ray, B., Posnett, D. Filkov, V., Devanbu, P. (2014). A large scale study of programming languages and code quality in GitHub. In: ACM SIGSOFT symposium on the foundations of software engineering, pp 155–65

[12] Ni, Z., Li, B., Sun, X., Chen, T., Tang, B., & Shi, X. (2020). Analyzing bug fx for automatic bug cause classifcation. Journal of Systems and Software, 163, 110538.

[13] Goseva-Popstojanova, K., Tyo, J. (2018). Identifcation of security related bug reports via text mining using supervised and unsupervised classifcation. In: Int. conf. on software quality, reliability and security, pp. 344–355.

[14] Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., & Mu, D. (2021). Improving high-impact bug report prediction with combination of interactive machine learning and active learning. Information and Software Technology, 133, 106530.

[15] Xia, X., Lo, D., Wang, X., Zhou, B. (2014). Automatic defect categorization based on fault triggering conditions. In: Int. conference on engineering of complex computer systems, pp. 39–48.

[16] Du, X., Zhou, Z., Yin, B., & Xiao, G. (2020). Cross-project bug type prediction based on transfer learning. Software Quality Journal, 28(1), 39–57.

[17] Ahmed, H. A., Bawany, N. Z., & Shamsi, J. A. (2021). Capbug-A framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. IEEE Access, 9, 50496–50512.

[18] Hammouri, A., Hammad, M., Alnabhan, M., & Alsarayrah, F. (2018). Software bug prediction using machine learning approach. International Journal of Advanced Computer Science and Applications, 9(2), 78–83.

[19] Wang, S., Wang, J., Nam, J. & Nagappan, N. (2021). Continuous software bug prediction. In Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (pp. 1–12).

[20] Khan, F., Kanwal, S., Alamri, S., & Mumtaz, B. (2020). Hyper-parameter optimization of classifers, using an artifcial immune network and its application to software bug prediction. IEEE Access, 8, 20954–20964.

[21] Gupta, D. L., & Saxena, K. (2017). Software bug prediction using object-oriented metrics. Sādhanā, 42(5), 655–669.

[22] Moustafa, S., ElNainay, M. Y., El Makky, N., & Abougabal, M. S. (2018). Software bug prediction using weighted majority voting techniques. Alexandria engineering journal, 57(4), 2763–2774.

[23] Qu, Y., & Yin, H. (2021). Evaluating network embedding techniques' performances in software bug prediction. Empirical Software Engineering, 26(4), 1–44.

[24] Yang, Y., Xia, X., Lo, D., Grundy, J. (2022). A survey on deep learning for software engineering. ACM Computing Surveys (CSUR), 54(10), 1–73.

[25] Huang, Q., Xia, X., Lo, D., & Murphy, G. C. (2020). Automating intention mining. IEEE Transactions on Software Engineering, 46(10), 1098–1119.

[26] Mahajan, G., & Chaudhary, N. (2022). Design and development of novel hybrid optimization-based convolutional neural network for software bug localization. Soft Computing, 26(24), 13651–13672.

[27] Rani, G., Pandey, U., Wagde, A. A., & Dhaka, V. S. (2022). A deep reinforcement learning technique for bug detection in video games. International Journal of Information Technology, 15(1), 355–367.

[28] Wang, Z., Tong, W., Li, P., Ye, G., Chen, H., Gong, X., & Tang, Z. (2023). BugPre: an intelligent software version-to-version bug prediction system using graph convolutional neural networks. Complex & Intelligent Systems, 9(4), 3835–3855.

[29] Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ragkhitwetsagul, C., & Ghose, A. (2021). Automatically recommending components for issue reports using deep learning. Empirical Software Engineering, 26(2), 1–39.

[30] Cynthia, S.T., Banani, R., & Debajyoti, M. (2022). Feature transformation for improved software bug detection models. In 15th Innovations in Software Engineering Conference, pp. 1–10

[31] Giray, G., Kwabena, E. B., Ömer, K., Önder, B., & Bedir, T. (2023). On the use of deep learning in software defect prediction. Journal of Systems and Software, 195, 111537.

[32] Xuewu, Z. H. A. O., Hongmei, W. A. N. G., Chaohui, L. I. U., Lingling, L. I., Shukui, B. O., & Junzhong, J. I. (2022). Artifcial jellyfsh search optimization algorithm for human brain functional parcellation. Journal of Frontiers of Computer Science & Technology, 16(8), 1829–1841.