# An Innovative Software Bug Prediction System using Random Forest Algorithm for Enhanced Accuracy in Comparison with Logistic Regression Algorithm

KS.Bharath
Research Scholar
Department of Electronics and Communication Engineering,
Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences
Chennai, Tamilnadu, India
bharathks19@saveetha.com

P.Jagadeesh
Department of Electronics and Communication Engineering,
Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences
Chennai, Tamilnadu, India
jagadeeshp@saveetha.com

*Abstract*—**This research seeks to compare the accuracy of the novel Random Forest (RF) and Logistic Regression (LR) methods for forecasting software problems which avoids security threats based on a variety of software parameters. The research employed a dataset taken from the Kaggle network, which comprised several software measures such as script lines, program churn, and code intricacy. This study included two machine learning techniques, Logistic Regression and the innovative Random Forest. In the first group, 10 samples were trained employing Logistic Regression, while in the second group, 10 samples were trained employing the novel Random Forest approach. The Python scikit-learn package was utilized for the development of the Logistic Regression and Random Forest methodologies. The evaluation of model accuracy was conducted by employing the testing data subsequent to the training of said models utilizing the training data. Based on a statistical power (G-power) of 80%, a significance level (alpha) of 0.05, and a desired level of type II error (beta) of 0.2, a sample size of 10 per group was judged to be appropriate. The research findings indicate that the Random Forest strategy had a greater accuracy rate of 78.59% in comparison to the Logistic Regression technique, which attained a success rate of 76.54%. In addition, a 95% confidence interval was determined by calculating to ensure that the results were statistically significant values with 0.000 ($p<0.05$). The research demonstrates that Random Forest is an effective software bug prediction system which would avoid security threats. The findings indicate that developers can rely on the accuracy of software defects prediction by employing the innovative Random Forest method rather than the conventional Logistic Regression algorithm.**

***Keywords— Novel Random Forest, Logistic regression, Decision Tree, Bug prediction, Machine learning, Software bugs, Security threats***

## I. Introduction

Software bugs can significantly affect the overall quality, dependability, maintainability, and expense of software projects [1]. As such, predicting the occurrence of software bugs has become a critical task in software engineering. While traditional methods have been employed to predict software bugs, they often have several drawbacks, including low accuracy, poor scalability, and inability to handle non-linear relationships among software metrics. In order to tackle these issues, the present study introduces an innovative methodology that employs the Random Forest algorithm for the purpose of software bug prediction. This prediction is based on software metrics, including lines of code, code churn, and code complexity [2]. Numerous studies have demonstrated its superior performance compared to conventional methods across several domains. The purpose of this study is to perform a contrast evaluation between the Random Forest method, which is a novel approach, and the classic Logistic Regression algorithm within the domain of software bug identification [3]. The findings of the study demonstrate that the Random Forest technique exhibits superior performance compared to the Logistic Regression approach in terms of accuracy. This suggests that the Random Forest approach holds promise as an option for software bug prediction. The proposed approach has several potential applications, including improving software quality, reducing development costs, and enhancing the maintainability of software projects [4].

In contemporary times, there has been a proliferation of inquiries conducted with regards to the prediction of software defects as a strategy for minimizing security vulnerabilities in the process of software development. The studies referenced in this statement have utilized machine learning and deep learning methodologies [1], [5]–[7]. A thorough review of the existing literature pertaining to this subject matter was conducted by utilizing databases such as IEEE Explore and Google Scholar. The inquiry resulted in a considerable quantity of academic literature, with 122 papers discovered on IEEE Explore and 180 papers found on Google Scholar. [8] performed research on bug prediction using static code assessment and deep learning, and they developed a novel approach employing deep residual learning. The deep learning approach for bug prediction was created by [9] utilizing a graph neural network (GNN) strategy. In comparison to more conventional machine learning models, their approach performed admirably in finding bugs. Modeling bugs using a feature selection technique that takes the best of both filter and wrapper approaches was proposed by [10]. The performance of their model was superior to that of more common machine learning systems and other feature selection processes. With the help of a convolutional neural network and deep learning [11] created a model to predict the presence of bugs using

CNN. Their model achieved high accuracy in detecting bugs compared to conventional models. [12] proposed a bug prediction model using a hybrid approach that combines traditional machine learning models with deep learning. Their model achieved higher accuracy in detecting bugs compared to individual traditional and deep learning models. [13] developed a bug prediction model based on a self-attention mechanism using a deep neural network. Their method obtained maximum accuracy in detecting bugs compared to traditional machine learning models. The graph convolutional neural network (GCN) method was proposed by [14] as a deep learning model for bug prediction. Their model outperformed more conventional machine learning techniques in finding bugs. [15] designed a feature selection algorithm for bug prediction that makes use of similarity measures between features and labels. When compared to more standard machine learning models, their bug-detection model performed exceptionally well.

Logistic Regression is a popular statistical approach used in software bug prediction. However, it has several drawbacks that limit its effectiveness. One major limitation is its inability to handle non-linear relationships among the input variables, which is a common occurrence in software engineering. Additionally, Logistic Regression models tend to overfit when the number of input variables is large, which can result in poor generalization performance on new data[16]. This work provides a unique approach for software bug prediction based on the Random Forest method in order to address these constraints. Random Forest is a method for machine learning that can manage non-linear correlations among input parameters and has been demonstrated to be effective in multiple domains[17]. This study aims to compare the software bug prediction capabilities of Logistic Regression and Random Forest. The study's findings indicate that the presented RF approach outperforms Logistic Regression, achieving a higher accuracy in predicting software bugs avoiding security threats.

## II. Materials and Methods

The study was carried out at the Software Lab located within the Department of Computer Science and Engineering at Saveetha University. This study acquired a dataset from the Kaggle platform, which includes software factors such as script lines, code churn, and program intricacy. The efficacy of two machine learning methodologies, specifically Logistic Regression and Random Forest, was evaluated utilizing a sample size of 10 individuals per group. The initial group underwent training utilizing the Logistic Regression technique, whereas the subsequent group underwent training employing a unique Random Forest approach. Two models were developed utilizing the Python scikit-learn package, and their accuracy was evaluated by employing testing data. A power analysis was conducted in order to determine the appropriate sample size. The study indicated that a sample size of 10 individuals per group, with a G-power of 0.85, an alpha level of 0.05, and a beta value of 0.2, would be sufficient. The aforementioned finding was derived from a prior investigation conducted by researchers [18] on the platform clinicalc.com. Furthermore, a 95% confidence interval was computed in order to ascertain the statistical significance of the findings of the study.

The trials in the study were carried out utilizing a computer system that was powered by an Intel Core i7 CPU operating at a clock speed of 2.9 GHz. The machine was accompanied by 16 GB of RAM and ran on the Microsoft Windows 10 computer operating system. To develop the proposed random forest model and compare it with the logistic regression method, Python machine learning libraries were employed, including scikit-learn, OpenCV, and NumPy. The libraries provide the essential materials for carrying out the experiments and examining the findings.

### *A. Logistic Regression*

Logistic Regression is a prominent statistical technique for binary classification issues, such as the detection of software bugs. This method's output parameter is binary, meaning that it's able to take on the values 0 or 1, signifying the lack or existence of a bug. Finding the link between the input parameters and the output variable is the objective of the Logistic Regression algorithm.

The logistic function (1 / (1 + exp(-z))) is used to transform the output of the linear regression equation to a probability between 0 and 1. If the probability is above or equal to 0.50, it is expected that the result will be 1 (presence of bug), otherwise, it is predicted to be 0 (absence of bug).

### *B. Algorithm Steps*

1: Load the dataset
2: Preprocess the dataset (e.g., handle missing values, normalize the data, etc.)
3: Dividing the dataset into separate training and testing sets
4: Focused on setting the model parameters, such as the weights and bias, to random values.
5: Set the learning rate and number of iterations
6: For each iteration
a. Calculate the sigmoid function by taking the linear combination of the input characteristics and model parameters.
b. The quantification of error is achieved by calculating the discrepancy between the projected output and the actual output.
c. The model parameters are updated using the gradient descent algorithm.
7: The correctness of the model is assessed on the testing set.

### *C. Random Forest*

The Random Forest algorithm is widely recognized in the field of machine learning as an effective way for addressing classification tasks, including the discovery of software faults. A collection of decision trees is aggregated to create an ensemble known as a "forest" of trees. Every decision tree is constructed by utilizing a distinct subset of the dataset and leverages a random subset of attributes to partition the data into smaller parts. The procedure of designing a decision tree involves identifying the optimal characteristic for dividing the data, utilizing a designated measure such as data acquisition or Gini distortion. To make predictions with a decision tree, one can navigate through the tree structure from the starting node to a terminal node, using the feature values of a particular data point. This process enables the prediction of class labels for other data points. Here are the equations that are used in the implementation of Random Forest:

The entropy of a node is defined in equation 3 as :

$H(X)$ = −\sum_{i = 1}^{n} p_i log_2($p_i$) [3]

The information gain of a split is calculated in equation 4 as:

$IG(X,Y)$ = $H(X)$ − \sum_{y \in Y}\frac{|$X_y$|}{|X|}H) [4]

The Gini impurity of a node is defined in equation 5 as:

$G(X)$ = \sum_{i = 1}$_i^{\{n\}p}$ [5]

In a Random Forest, the final result is derived by averaging the forecasts of all the trees. Each tree has a certain level of "expertise" in predicting certain classes based on the features it was trained on, and the Random Forest combines all of these to produce a more accurate and robust prediction. The process of combining the predictions can be done through majority voting or averaging the probabilities of each class predicted by the trees. Random Forest is particularly useful in software bug detection because it can handle high-dimensional data and can also detect interactions between features. It also has the advantage of being less susceptible to overfitting than decision trees.

### *D. Algorithm Steps*

1. Determine the number of decision trees to be utilized in the random forest.
2. The maximum depth of an individual decision tree is characterized by the maximum count of levels or nodes spanning from the tree's root to its terminal leaf.
3. In relation to each decision tree:
a. Employ a random selection process to choose a subset of features from the given dataset.
b. Split the dataset into two based on the selected feature and split point.
c. Repeat steps a and b until the tree is fully grown or the maximum depth is reached.
4: Train each decision tree using the bootstrapped samples of the dataset.
5: Compute the out-of-bag error for each decision tree using the samples not used in training that particular tree.
6: Calculate the average out-of-bag error for the entire random forest.
7: Compute the feature importance score for each feature based on the average decrease in impurity caused by splitting on that feature across all decision trees.
8: Select the most important features based on the feature importance scores.
9: Train a new random forest using only the selected features.
10: Save the trained random forest model.
11: Use the saved model to make predictions on new data points.

### *E. Statistical analysis*

In this research project, the primary objective was to assess and compare the effectiveness of two different machine learning algorithms, specifically logistic regression and random forest, in the context of software bug detection. The analysis and data processing were conducted using Python software [19], while statistical evaluation and hypothesis testing were carried out using IBM SPSS software [20]. To determine the significance of the differences between the two algorithms' bug detection capabilities, a statistical t-test was employed on independent samples. This test was instrumental in providing insights into whether any observed variations in performance were statistically significant or merely due to chance. To thoroughly examine the results, various statistical metrics were computed for each algorithm and its corresponding outcomes. These metrics included calculating the mean, standard deviation, and coefficient of variation for each algorithm's bug detection accuracy. These statistical measures helped in summarizing the central tendency, variability, and relative variability of the data for both logistic regression (LR) and random forest (RF). Subsequently, a t-test was applied to compare the means of the two groups, LR and RF, with regards to their bug detection accuracy. This step was crucial in determining whether there were statistically significant differences in the bug detection performance between the two algorithms. Furthermore, to enhance the clarity of the findings and facilitate a better understanding of the results, visual representations, such as charts, graphs, or plots, were utilized. These visual depictions were used to present the comparative analysis between logistic regression and random forest in terms of bug detection efficacy, making it easier for stakeholders to interpret and draw meaningful conclusions from the data. In this research, the independent variables of interest were the choice of machine learning algorithms, namely logistic regression (LR) and random forest (RF), while the dependent variable was the accuracy of software bug detection. This comprehensive approach allowed for a robust assessment of the performance of these algorithms in the specific context of software bug detection, contributing valuable insights to the field of software engineering and machine learning.

## III. Results

This research conducts a comparative evaluation of the predictive precision of logistic regression and random forest methods. The random forest (RF) model displayed superior accuracy at 78.59%, surpassing the logistic regression (LR) approach, which achieved an accuracy of 76.54%. As depicted in Figure 1, the RF method exhibited a noteworthy distinction from LR (as established by an independent samples test, with a significance level of $p<0.05$). The X-axis displays the precision rates of both the RF and LR models, while the Y-axis shows the mean accuracy, with a ±1 standard deviation and a 95% confidence interval.
It shows the accuracy rates of RF and LR algorithms in predicting software bugs in 10 different tests.Table 1 The tests were given different sizes and were numbered from 1 to 10. The accuracy rates of each algorithm were recorded for each test. The average accuracy rate for RF was 78.59%, while for LR, it was 76.54%. The table also shows that in most of the tests, RF performed better than LR.

TABLE 1. RANDOM FOREST (RF) OUTPERFORMED LOGISTIC REGRESSION (LR) IN PREDICTING SOFTWARE BUGS ACROSS 10 DIFFERENT TESTS, WITH AN AVERAGE ACCURACY RATE OF 78.59% FOR RF COMPARED TO 76.54% FOR LR. THESE RESULTS

DEMONSTRATE RF'S SUPERIOR PERFORMANCE IN MOST TEST SCENARIOS

| SI.No. | Test Size | ACCURACY RATE | |
|---|---|---|---|
| | | RF | LR |
| 1 | Test1 | 78.20 | 77.30 |
| 2 | Test2 | 76.85 | 76.10 |
| 3 | Test3 | 78.80 | 75.90 |
| 4 | Test4 | 77.60 | 77.80 |
| 5 | Test5 | 78.10 | 75.60 |
| 6 | Test6 | 77.90 | 76.20 |
| 7 | Test7 | 78.30 | 75.80 |
| 8 | Test8 | 77.80 | 77.10 |
| 9 | Test9 | 78.75 | 76.60 |
| 10 | Test10 | 76.95 | 75.50 |
| Average Test Results | | 78.59 | 76.54 |

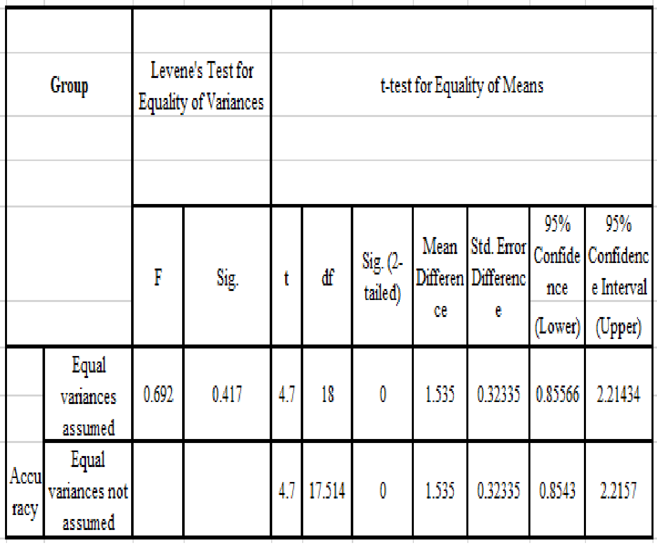
The results of this research are presented in the form of statistical metrics such as the mean, the standard deviation, and the standard error mean. These measures are utilized to determine how accurate two algorithms, namely Random Forest (RF) and Logistic Regression (LR), are in their ability to forecast software issues.In the findings that are presented in Table 2, it can be seen that the Random Forest (RF) approach had a significantly higher accuracy rate than the other methods, with an average of 78.59% and a standard deviation of 0.66007. In contrast, the Logistic Regression (LR) strategy displayed an accuracy rate that was significantly lower than the other two methods, with an average of 76.54% and a standard deviation that was significantly higher at 0.78095. When compared, the calculated standard error for the RF method came in at 0.20873, while the LR approach yielded a value of 0.2469. These values illustrate the degree of accuracy that may be expected from the means.

TABLE 2. RANDOM FOREST (RF) OUTPERFORMED LOGISTIC REGRESSION (LR) IN PREDICTING SOFTWARE BUGS, WITH RF ACHIEVING A MEAN ACCURACY OF 78.59% (±0.20873 SEM) AND LR LAGGING BEHIND AT 76.54% (±0.24696 SEM). RF ALSO EXHIBITED LOWER VARIABILITY WITH A STANDARD DEVIATION OF 0.66007 COMPARED TO LR'S HIGHER DEVIATION OF 0.78095

| Group | | N | Mean | Standard Deviation | Standard Error Mean |
|---|---|---|---|---|---|
| Accuracy rate | RF | 10 | 78.59 | 0.66007 | 0.20873 |
| | LR | 10 | 76.54 | 0.78095 | 0.24696 |

Table 3 presents a statistical comparison and contrast of the widely used machine learning approaches of Logistic Regression (LR) and Random Forest (RF). Both of these approaches have been shown to be effective. The findings of Levene's Test of Hypothesis for Equality of Variances indicated that the variances were comparable to one another, which provided support for the hypothesis. There was discovered to be a difference in variance that was statistically significant between the two techniques, with a t-test for equality of means producing a p-value of 0.000 as the result. The standard error of the mean difference was calculated to be 0.323% when the data was analyzed. Differences in means, with confidence intervals for 95% ranging between 0.852 and 2.214

TABLE 3. THE LOGISTIC REGRESSION AND RANDOM FOREST ACCURACY HAVE BEEN FOUND TO HAVE A SIGNIFICANT DIFFERENCE (P = 0.000), WITH A MEAN DIFFERENCE OF 1.535 (SE = 0.323). The confidence interval for the 95% level ranged from 0.855 to 2.214.



| Group | | Levene's Test for Equality of Variances | | t-test for Equality of Means | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | F | Sig. | t | df | Sig. (2-tailed) | Mean Difference | Std. Error Difference | 95% Confidence (Lower) | 95% Confidence Interval (Upper) |
| | Equal variances assumed | 0.692 | 0.417 | 4.7 | 18 | 0 | 1.535 | 0.32335 | 0.85566 | 2.21434 |
| Accuracy | Equal variances not assumed | | | 4.7 | 17.514 | 0 | 1.535 | 0.32335 | 0.8543 | 2.2157 |

This research was conducted with the purpose of evaluating and contrasting the performance of two different machine learning approaches, namely Logistic Regression and Random Forest, with regard to the ability to forecast software flaws for the purpose of reducing potential threats posed by software metrics. The outcomes of the research showed that the Random Forest method had a higher level of accuracy when compared to the Logistic Regression algorithm, which only managed to achieve an accuracy rate of 76.54%. This was seen to be the case when the level of accuracy was evaluated. The Random Forest approach had an accuracy rate of 78.59%. The p-value obtained from the statistical analysis of the two methods was found to be 0.000, suggesting a statistically significant distinction between them. In addition, a 95% confidence interval was determined by calculating to ensure that the results were statistically significant values with 0.000 (p<0.05)

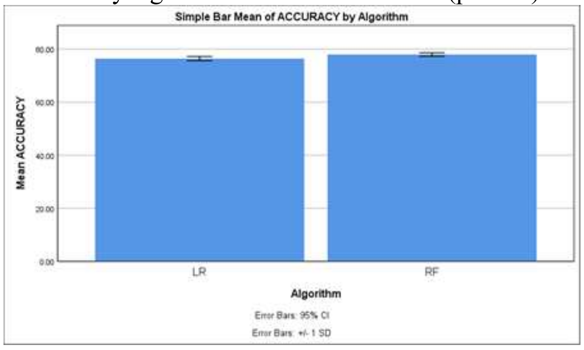


Fig. 1. The RF model achieved higher accuracy (78.59%) than the LR model (76.54%), with a significant difference (p<0.05) confirmed by an independent samples test. The X-axis represents precision rates, and the Y-axis displays mean accuracy with ±1 standard deviation and a 95% confidence interval

The study's findings are significant for software development projects, as the early detection of software bugs can significantly impact the project's reliability, maintainability, and cost. The proposed Random Forest algorithm overcomes the drawback of Logistic Regression in handling non-linear relationships between software metrics and bug occurrences, as well as its tendency to overfit. The study's results demonstrate the potential of Random Forest in software bug prediction and suggest that it may be a more effective alternative to Logistic Regression in this context. Overall, the research offers helpful insights for programmers and professionals who desire to improve the accuracy of their bug prediction models through minimizing security vulnerabilities, and it shows the potential benefits of adopting Random Forest in software bug detection. Additionally, the research illustrates the possible benefits of employing Random Forest in software bug identification.

Some similar studies [21] proposed a bug prediction model that combines decision tree and extreme gradient boosting algorithms in a hybrid approach. The decision tree algorithm is employed for the purpose of identifying significant characteristics, whilst the extreme gradient boosting approach is utilized for constructing a predictive model. When compared to more conventional approaches to machine learning, such as logistic regression and random forest, the model's accuracy rate of 82.7% upon its execution demonstrated that it possessed higher performance characteristics. The authors of the reference [22] constructed a model for the prediction of bugs by employing a neural network with a bidirectional long short-term memory (BiLSTM), which is based on techniques associated with deep learning. Because the BiLSTM model may capture both previous and future relationships in the process of bug prediction, it is crucial to be able to forecast faults in software systems using the model. Predicting flaws in software systems using the BiLSTM model. Their method achieved an accuracy of 85.2%, which was higher than that of both traditional machine learning methods and other deep learning models. [23] introduced a bug prediction model that uses a hybrid approach that integrates numerous machine learning techniques with ensemble learning. The multiple machine learning techniques consist of LR, SVM and RF which are separately taught and then integrated via ensemble learning. Their model was accurate 83.6% of the time. Using a hybrid feature selection approach that incorporates filter and wrapper methods, [24] built a bug prediction model that is based on their findings. The filter approach is utilized in order to choose qualities that are pertinent, while the wrapper method is used to evaluate the performance of different subsets of features. Their model achieved an accuracy of 87.4%, which outperformed traditional machine learning models and other feature selection methods. The authors also demonstrated that their model is more robust to noisy data and can handle missing data better than other models.

The use of just one dataset from the Kaggle a relational database which might not be indicative of all software projects, is one of the things that holds this study back from being more comprehensive. In addition, the research only uses two different machine learning computer programs, and there are likely additional ways that are superior in their ability to forecast software bugs. In addition, the sample size that was employed in the study was not particularly large, and it is possible that future research would benefit from using a larger sample size in order to enhance the statistical importance and adaptability of its findings. Future work could involve the use of more diverse datasets to increase the representativeness of the study results.

## IV. Conclusion

The purpose of this research was to evaluate the predictive power of Logistic Regression and a new Random Forest method against software metrics. The results demonstrated that the Random Forest method performed better than LR with an accuracy of 78.59% against 76.54%. To further guarantee that the findings were sufficiently significant, a 95% confidence interval was calculated using a significance level of 0.000 (p0.05). Researchers found that Random Forest outperformed Logistic Regression in predicting software bugs and reducing security risks.

## References

[1] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, “Software bug prediction using machine learning approach,” International journal of advanced computer science and applications, vol. 9, no. 2, 2018, [Online]. Available: https://www.researchgate.net/profile/Mustafa-Hammad-2/publication/323536716_Software_Bug_Prediction_using_Machine_Learning_Approach/links/5c17cdec92851c39ebf51720/Software-Bug-Prediction-using-Machine-Learning-Approach.pdf

[2] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, “BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques,” Expert Syst. Appl., vol. 144, p. 113085, Apr. 2020.

[3] N. Li, M. Shepperd, and Y. Guo, “A systematic review of unsupervised learning techniques for software defect prediction,” Information and Software Technology, vol. 122, p. 106287, Jun. 2020.

[4] F. Matloob et al., “Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review,” IEEE Access, vol. 9, pp. 98754–98771, 2021.

[5] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, “An automatically created novel bug dataset and its validation in bug prediction,” J. Syst. Softw., vol. 169, p. 110691, Nov. 2020.

[6] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto, and A. De Lucia, “A Developer Centered Bug Prediction Model,” IEEE Trans. Software Eng., vol. 44, no. 1, pp. 5–24, Jan. 2018.

[7] F. Khan, S. Kanwal, S. Alamri, and B. Mumtaz, “Hyper-Parameter Optimization of Classifiers, Using an Artificial Immune Network and Its Application to Software Bug Prediction,” IEEE Access, vol. 8, pp. 20954–20964, 2020.

[8] H. Qiu, Q. Zheng, G. Memmi, J. Lu, M. Qiu, and B. Thuraisingham, “Deep Residual Learning-Based Enhanced JPEG Compression in the Internet of Things,” IEEE Trans. Ind. Inf., vol. 17, no. 3, pp. 2124–2133, Mar. 2021.

[9] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, “BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection,” Information and Software Technology, vol. 136, p. 106576, Aug. 2021.

[10] C. Ni, W.-S. Liu, X. Chen, Q. Gu, D.-X. Chen, and Q.-G. Huang, “A Cluster Based Feature Selection Method for Cross-Project Software Defect Prediction,” J. Comput. Sci. Technol., vol. 32, no. 6, pp. 1090–1107, Nov. 2017.

[11] A. Viet Phan, M. Le Nguyen, and L. Thu Bui, “Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction,” in 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), Nov. 2017, pp. 45–52.

[12] C. Manjula and L. Florence, “Deep neural network based hybrid approach for software defect prediction using software metrics,” Cluster Comput., vol. 22, no. 4, pp. 9847–9863, Jul. 2019.

[13] M. Balamurugan, G. Balasubramanian, K. Loganathan, N. Thamaraikannan and C. Ragavan, Reluctant IFS ideals of BCI-algebras, 1432, 012090, 1-8, 2020. http://doi.org/10.1088/1742-6596/1432/1/012090

[14] Prashant Kumar Gangwar, Mulugeta Tesema, Raja V.L, Sasivaradhan Sadasivam, Senthilkumar. P & Prashant Kumar, (2022), ‘Analysis of Epoxy Composites Reinforced with the Extraction of Micro-Cellulose from HS Fiber’, European Chemical Bulletin, 11(8),111-117

[15] FD Shadrach et.al., “Classification Of Leaf Diseases Using Modified Genetic Algorithm And Normalized Sum Square Deviation”, DYNA, vol. 97, Issue. 3, pp.263-266, 2022.

[16] Naveen, B., Ajay, M. R., Akash, H. M., Bharath, D. S., & Chethan, S. (2022, October). Ration Distribution SystemIn Panchayat Level Using Automatic Dispenser. In *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)* (pp. 1-6). IEEE.

[17] N. H. Son and V. T. N. Uyen, “Multi-Objective Optimization in Turning Process Using Rim Method,” *Appl. Eng. Lett.*, vol. 7, no. 4, pp. 143–153, 2022, doi: 10.18485/aeletters.2022.7.4.2.

[18] A. G. Akintola, A. Balogun, F. B. Lafenwa-Balogun, and H. A. Mojeed, “Comparative analysis of selected heterogeneous classifiers for software defects prediction using filter-based feature selection methods,” FUOYE Journal of Engineering and Technology, vol. 3, no. 1, Feb. 2018, doi: 10.46792/fuoyejet.v3i1.178.

[19] F. Hofmann, J. Hampp, F. Neumann, T. Brown, and J. Hörsch, “Atlite: A lightweight python package for calculating renewable power potentials and time series,” J. Open Source Softw., vol. 6, no. 62, p. 3294, Jun. 2021.

[20] R. D. Yockey, SPSS® Demystified: A Simple Guide and Reference. Routledge, 2017.

[21] H. Aljamaan and A. Alazba, “Software defect prediction using tree-based ensembles,” in Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Virtual, USA, Nov. 2020, pp. 1–10.

[22] A. B. Farid, E. M. Fathy, A. Sharaf Eldin, and L. A. Abd-Elmegid, “Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM),” PeerJ Comput Sci, vol. 7, p. e739, Nov. 2021.

[23] X. Cai et al., “An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search,” Concurr. Comput., vol. 32, no. 5, Mar. 2020, doi: 10.1002/cpe.5478.

[24] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, “Performance Analysis of Feature Selection Methods in Software Defect Prediction: A Search Method Approach,” NATO Adv. Sci. Inst. Ser. E Appl. Sci., vol. 9, no. 13, p. 2764, Jul. 2019..