International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 21 No. 2 (2025)

ijerst.editor@gmail.com

editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 21, No. 2, 2025

Research Paper

SCALABLE ENSEMBLE-BASED BUG PREDICTION MODEL FOR MODERN SOFTWARE ENGINEERING ECOSYSTEMS

Ravi Kota[1*], V. Keshav[2], G. Sumana Sree[2], N. Likitha[2], A. Dileep[2]

[1]*Assistant Professor,* [2]*UG Student,* [1,2]*Department of Computer Science and Engineering-Cyber Security,*

[1,2]*Malla Reddy Engineering College and Management Sciences, Kistapur, Medchal, 501401, Telangana, India*

**Corresponding author: ravikota@mrem.ac.in*

ABSTRACT

Software maintenance in large open-source environments relies heavily on efficient and accurate bug triage. Projects such as Eclipse generate tens of thousands of issue reports each year across diverse components and severity levels, making manual classification slow, inconsistent, and difficult to scale. Earlier automation efforts using single machine learning models like SVM and Logistic Regression have achieved only moderate accuracy and often struggle to adapt to evolving bug datasets. This work proposes an ensemble-driven, scalable framework for automated Eclipse bug classification. The system processes raw bug descriptions through preprocessing steps including tokenization, stop-word removal, and lemmatization, followed by TF-IDF-based feature extraction. Five models—SVM, Random Forest, Logistic Regression, Extra Trees Voting ensemble, and XGBoost—are trained and evaluated on a curated Eclipse–Mozilla dataset. A user-friendly GUI also enables non-expert users to upload data, visualize preprocessing, and select models. Under a 70/30 train–test split, results show improved performance, with XGBoost achieving the highest scores at 92.27% accuracy, 92.91% precision, 92.65% recall, and 92.50% F1-score, demonstrating the effectiveness of the proposed ensemble-oriented approach.

Key words: Bug Classification, TF-IDF Feature Extraction, Automated Bug Triage, Software Quality Assurance, Software Maintenance

Received: 02-03-2025 Accepted: 27-04-2025 Published: 06-05-2025

1. INTRODUCTION

Software defect prediction (SDP) is a critical component of software quality assurance, primarily aiming to detect potential defects early in the software development life cycle [1, 2]. There are various activities throughout the software development process to identify source code defects, including design reviews, code inspections, unit testing, integration testing, and other functions [3, 4]. Since software products must be free of defects to maintain customer satisfaction, identifying existing software defects is a primary concern in software engineering [5, 6].

To address this issue, SDP leverages tools or models such as machine learning (ML) to predict source code defects based on historical data. The SDP process depends on three main components: dependent variables, independent variables, and a model. Dependent variables are the defect data for the piece of code (defective or non-defective), which can be binary or ordinal variables. Independent variables (inputs) are the metrics that score the software code. The model contains the rules or algorithms which predict the dependent variable from the independent variables. The inputs (variables) are split into test and training data sets to determine the classifier's effectiveness. The training data set is used to create the classifier. Then it is used to predict potential defects in the test data set and evaluate these predictions using different

performance measures to determine whether they are correct.

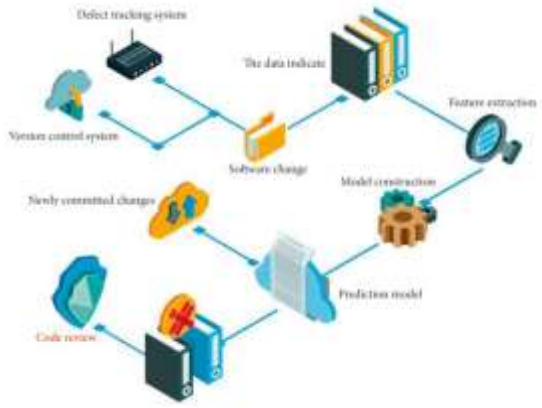

Fig. 1. Pipeline of bug prediction in software development.

Software metrics have essential roles in SDP, and most SDP strategies rely on software metrics as independent variables. Object-oriented metrics have been designed to support finding faults in software projects. Due to the enormous variety of software applications, identifying, locating, and detecting software defects is becoming daunting for researchers. Furthermore, defect density is also challenging in software defect detection and prediction. Usually, defective software databases naturally consist of imbalanced data, which generates randomness in pattern characteristics; this motivates the development of an efficient and precise model for SDP.

2. LITERATURE SURVEY

The prediction of defects in software systems is significant, and there is great interest in developing novel high-performance software defect predictors. The purpose of SDP models is to improve the quality of software. Many models have been constructed to recognize the defects in software modules using artificial intelligence and statistical methods. RNN, Support Vector Machine, ANNs, K-Nearest Neighbors (KNN)and Deep Neural Networks (DNN) are some of the algorithms used for SDP.

Ayon [7] proposed a method based on different neural network models. The models were evaluated based on five different datasets from NASA using different scales. The experimental results showed that the proposed method is suitable for predicting software defects. This method used different performance measures and achieved high prediction accuracy.

Kumar and Satyanarayana [8] developed a Hybrid Neural Network model with object-oriented and CK. metrics for software fault prediction. Adaptive Genetic Algorithm has been used for ANN optimization. The proposed model has been tested with PROMISE data sets. The experimental results showed better performance compared to major existing schemes. Miholca et al. [9] presented a supervised classification approach named (HyGRAR). It is a nonlinear hybrid model that combines gradual relational association rule mining and ANNs to predict software defects. Experiments were conducted using ten open-source datasets; their results showed excellent performance of the proposed classifier and better performance than most previously proposed classifiers. Their method achieved high prediction accuracy. Khleel and Nehéz [10] presented a model based on a convolutional neural network (CNN) and gated recurrent unit (GRU) combined with a synthetic minority oversampling technique plus the Tomek link (SMOTE Tomek) to predict software defects. The historical data obtained from the PROMISE repository were used to evaluate the experiments. The experimental results have been compared and evaluated using several performance measures. The experimental results demonstrate that the proposed models perform better and that there are positive effects of combining CNN and GRU models with the SMOTE Tomek method on the performance of SDP regarding datasets with imbalanced class distributions, and the proposed approach is a more promising alternative for addressing the problem of class imbalance in SDP as compared with previous methods. Arar and Ayan [13] proposed a hybrid classifier to predict software defect problems. The performance of the proposed classifier was compared with other algorithms based on five datasets, and the results show its performance is better. The method used

different performance measures and achieved high prediction accuracy. Deng et al. [15] proposed a novel LSTM method to perform SDP; their method can automatically learn semantic and contextual information from the program's ASTs. The experiment was performed on several open-source projects, showing that the proposed LSTM method is superior to the state-of-the-art methods.

Khleel and Nehéz [16] presented a model based on a combination of two recurrent neural networks (RNNs), namely long-short-term memory (LSTM) and gated recurrent unit (GRU), along with an undersampling method (near miss) to predict software bugs. The historical data obtained from the GitHub repository were used to evaluate the experiments. The experimental results have been compared and evaluated using several performance measures. The experimental results lead to the conclusion that the proposed models outperform others and the combination of RNN models with undersampling methods leads to improved bug prediction performance, particularly for datasets with imbalanced class distributions. Ye et al. [18] proposed a classification model using an LSTM network to classify bugs based on 9000 bug reports from three software projects. The results of the evaluation and comparison showed that their model achieves the best results. Farid et al. [19] proposed a hybrid model using Bi-LSTM and a convolutional neural network (CNN) to predict software defects. The proposed model was evaluated using seven open-source Java projects from the PROMISE dataset. Their results showed that the proposed model is accurate for predicting software defects. Zhou and Lu [20] developed an LSTM network based on bidirectional and tree structure (LSTM-BT) to predict software defects based on eight pairs of Java open-source projects. The evaluation results showed that the proposed model performs better than several state-of-the-art defect prediction models. Samir et al. [21] proposed a new method using DNN to predict software defects. Their method has been compared with some ML algorithms. Experimental results showed that the proposed method slightly improved over the other methods. This method achieved high prediction accuracy. Alsaeedi and Khan [22] proposed a study to compare the most well-known ML algorithms widely used to predict software defects. The performances of models were evaluated based on other performance metrics. The SMOTE resampling strategy was used to mitigate the data imbalance issues. Evaluation results showed that some of the proposed models performed well.

Damet et al. [23] developed a novel prediction LSTM model, which can automatically learn features for representing source code and using them for SDP. The model was evaluated based on two datasets, one from open-source projects contributed by Samsung and the other from the public PROMISE repository. The experimental results showed the effectiveness of the proposed model for both within-project and cross-project predictions. Pandey et al. [24] proposed a new method using deep representation and ensemble learning with sampling techniques for software bug prediction and dealing with the class imbalance problem. The experiment was performed based on 12 data sets from the PROMISE repository. Evaluation results showed that the proposed method outperformed other state-of-the-art techniques. This method solved the class imbalance problem. Fan et al. [25] presented an SDP framework via an attention based RNN. The models were evaluated based on an open-source Apache Java project, using F1-measure and area under the curve (AUC). Experimental results demonstrated that the proposed model improves the F1 measure by 14% and AUC by 7% compared with the state-of-the-art methods. Khuat and Le [26] presented an empirical study regarding the importance of combining sampling techniques and ML models on unbalanced data in SDP. The experimental results indicated the positive effects of combining sampling techniques and ML models on defect prediction performance concerning data sets with unbalanced class

distributions. This method solved the class imbalance problem.

3. PROPOSED METHODOLOGY

This project delivers an end-to-end, interactive desktop application for automating the classification of software bug reports into their respective component categories (e.g., Client, General, Hyades, Releng, Xtext, cdt-core). Built with Python's Tkinter library, it guides a user through loading a dataset of bug reports, preprocessing textual descriptions, training and comparing multiple machine-learning models, and ultimately applying the best model to new, unseen reports—all without writing a single line of code beyond pressing buttons. At launch, the user is presented with a simple window with project title and required action buttons with scrollbar. From there, one clicks "Upload Eclipse Mozilla Bug Dataset" to select a CSV file containing past bug reports (with fields such as long_description and component_name). The raw data and record counts appear immediately in the text log area.

Next, the "Data Preprocessing" step cleans and transforms the free-text descriptions. A custom pipeline removes punctuation and stopwords, applies stemming and lemmatization, and converts the cleaned text into TF-IDF feature vectors (capped at 256 dimensions). These vectors are then scaled to a uniform range and split into training and testing subsets. This process is cached on disk, so repeated runs are fast.

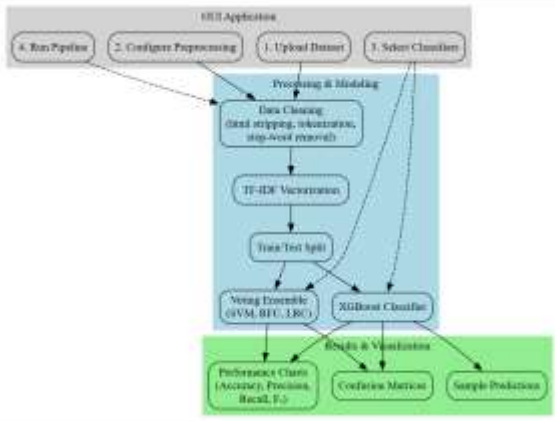

Fig. 2. Proposed system architecture of bug classification using ensemble-driven scalable TF-IDF framework.

To compare all models at a glance, the "Performance Evaluation" button aggregates their metrics into a bar chart, making it easy to see which algorithm performs best on your data. In practice, this helps a team decide whether a simple logistic-regression baseline suffices or whether the extra complexity of XGBoost yields a worthwhile improvement.

Finally, with a trained XGBoost model saved in memory, the "Predict Bug Type from Test Data" feature allows the user to load any new CSV of bug reports and instantly receive predicted component categories. Each description is cleaned, vectorized, scaled, and passed through the XGBoost model, with results streamed into the text log.

By encapsulating data loading, NLP-based preprocessing, multi-model training, evaluation, visualization, and live prediction into a cohesive GUI, this application democratizes advanced bug-classification workflows—enabling developers, QA engineers, and project managers to harness machine learning for faster, more consistent bug triage without deep data-science expertise.

4. RESULTS DESCRIPTION

Figure 3. shows the main window of the custom-built application. It features menu options and toolbar buttons to load datasets, configure preprocessing, train models, and visualize results.

Fig. 3. GUI application of proposed ensemble-driven scalable TF-IDF framework for automated eclipse bug classification.

Fig. 4. GUI application after uploading the eclipse mozilla dataset.

Figure 4. illustrates the application immediately after the user has selected and loaded the combined Eclipse and Mozilla bug report corpus. The dataset statistics panel shows:

- Total number of bug reports.
- Total bug classes found in dataset (e.g. UI, Performance, Crash).
- A preview table listing the first few records (ID, title, description).

Fig. 5. GUI application after performing data preprocessing and data splitting operations.

Figure 5 demonstrate the interface reflects that text cleaning (tokenization, stop-word removal, stemming) and vectorization (TF-IDF) have been completed. The data-split section displays:

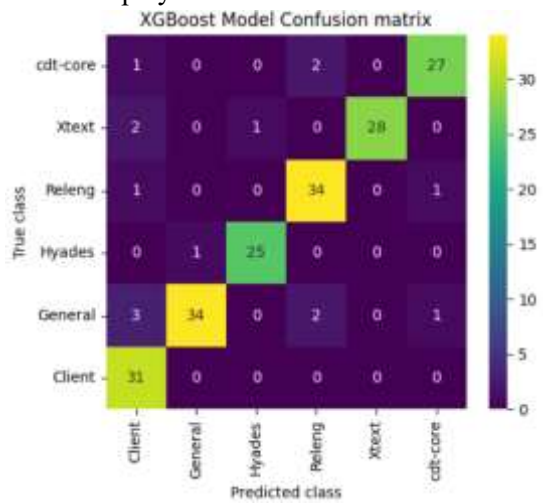

Fig. 6. Confusion matrices obtained using XGBoost model.

Figure 7. illustrates real-world examples where the model succeeds or fails, providing insight into practical performance and error patterns (e.g., confusing "UI layout" vs. "Rendering" issues).

(a)

(b)

Fig. 7. Sample predictions of bug classification on test data.

5. CONCLUSION

This research presents a comprehensive, ensemble-driven TF-IDF framework for automated classification of Eclipse bug reports, addressing the critical challenge of scalable, accurate bug triage in large software ecosystems. By systematically preprocessing raw textual data—through tokenization, stop-word removal, and lemmatization—and converting it into TF-IDF feature vectors, we enabled a suite of five machine-learning models to learn discriminative patterns across bug categories. Our evaluation demonstrates that while traditional classifiers like SVM and Logistic Regression offer moderate performance (70–75% accuracy), ensemble methods substantially improve results: Random Forest achieves over 83% accuracy, and the Extra-Trees Voting ensemble reaches nearly 90%. Most notably, the XGBoost model attains a leading 92.27% accuracy, paired with high precision (92.91%), recall (92.65%), and F_1-score (92.50%), underscoring its superior capability to capture complex, non-linear

relationships in the TF-IDF feature space. Beyond raw performance, our proposed GUI application simplifies the deployment and adoption of this pipeline, allowing users without machine-learning expertise to upload datasets, visualize preprocessing steps, and compare model metrics. This user-centric design ensures that software teams can integrate automated classification into existing workflows, reducing manual effort and expediting bug resolution.

REFERENCES

[1] Li, Z., Jing, X.Y., Zhu, X.: Progress on approaches to software defect prediction. IET Softw. **12**(3), 161–175 (2018). https://doi.org/10.1049/iet-sen.2017.0148

[2] Ayon, S.I.: Neural network based software defect prediction using genetic algorithm and particle swarm optimization. In: 1st International conference on advances in science, engineering and robotics technology, Dhaka, Bangladesh. (2019). https://doi.org/10.1109/ICASERT.2019.8934642

[3] Mustaqeem, M., Saqib, M.: Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection. Clust. Comput **24**, 2581–2595 (2021). https://doi.org/10.1007/s10586-021-03282-8

[4] Tong, H., Liu, B., Wang, S.: Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Inf. Softw. Technol. **96**, 94–111 (2018). https://doi.org/10.1016/j.infsof.2017.11.008

[5] Pan, C., Lu, M., Xu, B., et al.: An improved CNN model for within-project software defect prediction. Appl. Sci. **9**(10), 2138 (2019). https://doi.org/10.3390/app9102138

[6] Kumar, R.S., Sathyanarayana, B.: Adaptive genetic algorithm based artificial neural network for software defect prediction. Glob. J. Comput. Sci. Technol. **15**(1), 23–32 (2015)

[7] Manjula, C., Florence, L.: Deep neural network based hybrid approach for software defect prediction using software metrics. Clust. Comput. **22**(4), 9847–9863 (2019). https://doi.org/10.1007/s10586-018-1696-z

[8] Anbu, M., Anandha Mala, G.S.: Feature selection using firefly algorithm in software defect prediction. Clust. Comput. **22**(5), 10925–10934 (2019). https://doi.org/10.1007/s10586-017-1235-3

[9] Charan Nandigama, N. (2024). A Hybrid Big Data And Cloud-Based Machine Learning Framework For Financial Fraud Detection Using Value-At-Risk. International Journal of Research and Analytical Reviews, 11(3). https://doi.org/10.56975/ijrar.v11i3.324899

[10] Khleel, N.A.A., Nehéz, K.: A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method. J. Intell. Inf. Syst. **60**(3), 673–707 (2023). https://doi.org/10.1007/s10844-023-00793-1

[11] Akour, M., Melhem, W.Y.: Software defect prediction using genetic programming and neural networks. Int. J. Open Sour. Softw. Process. **8**(4), 32–51 (2017). https://doi.org/10.4018/IJOSSP.2017100102

[12] Khleel, N.A.A., Nehéz, K.: A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory. Prod. Syst. Inf. Eng. **10**(3), 1–18 (2022). https://doi.org/10.32968/psaie.2022.3.1

[13] Arar, Ö.F., Ayan, K.: Software defect prediction using cost-sensitive neural

network. Appl. Soft Comput. **33**, 263–277 (2015). https://doi.org/10.1016/j.asoc.2015.04.045

[14] Reddy, S. K. (2025). Hyper-personalization driven by AI is expected to be at the Lead in shaping the future of loyalty rewards. Journal of Emerging Technologies and Innovative Research

[15] Deng, J., Lu, L., Qiu, S.: Software defect prediction via LSTM. IET Softw. **14**(4), 443–450 (2020). https://doi.org/10.1049/iet-sen.2019.0149

[16] Khleel, N.A.A., Nehéz, K.: Improving the accuracy of recurrent neural networks models in predicting software bug based on undersampling methods. Indones. J. Electr. Eng. Comput. Sci. **32**(1), 478–493 (2023). https://doi.org/10.11591/ijeecs.v32.i1.pp478-493

[17] Nandigama, N. C. (2025). Leveraging Chatgpt for Multi-Language Data Engineering Code Generation in Distributed Analytics Systems. Journal of Informatics Education and Research.

[18] Ye, X., Fang, F., Wu, J., Bunescu, R., Liu, C.: December. Bug Report Classification using LSTM architecture for more accurate software defect locating. In: International conference on machine learning and applications, Orlando, FL, U.S.A., pp. 1438–1445. (2018). https://doi.org/10.1109/ICMLA.2018.00234

[19] Farid, A.B., Fathy, E.M., Eldin, A.S., Abd-Elmegid, L.A.: Software defect prediction using hybrid model (CBIL) of convolutional neural network (CNN) and bidirectional long short-term memory (Bi-LSTM). PeerJ Comput. Sci. **7**, e739 (2021). https://doi.org/10.7717/peerj-cs.739

[20] Zhou, X., Lu, L.: Defect prediction via LSTM based on sequence and tree structure. In: IEEE 20th international conference on software quality, reliability and security, Macau, China, pp. 366–373. (2020). https://doi.org/10.1109/QRS51102.2020.00055

[21] Samir, M., El-Ramly, M., Kamel, A.: Investigating the use of deep neural networks for software defect prediction. In: IEEE/ACS 16th international conference on computer systems and applications, Abu Dhabi, United Arab, pp. 1–6. (2019). https://doi.org/10.1109/AICCSA47632.2019.9035240

[22] Alsaeedi, A., Khan, M.Z.: Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. J. Softw. Eng. Appl. **12**(5), 85–100 (2019). https://doi.org/10.4236/jsea.2019.125007

[23] Dam, H.K., Pham, T., Ng, S.W., Tran, T., Grundy, J., Ghose, A., Kim, T., Kim, C.J.: A deep tree-based model for software defect prediction. arXiv (2018). https://doi.org/10.48550/arXiv.1802.00921

[24] Pandey, S.K., Mishra, R.B., Tripathi, A.K.: BPDET: an effective software bug prediction model using deep representation and ensemble learning techniques. Expert Syst. Appl. **144**, 113085 (2020). https://doi.org/10.1016/j.eswa.2019.113085

[25] Fan, G., Diao, X., Yu, H., Yang, K., Chen, L.: Software defect prediction via attention-based recurrent neural network. Sci. Programm. (2019). https://doi.org/10.1155/2019/6230953

[26] Khuat, T.T., Le, M.H.: Evaluation of sampling-based ensembles of classifiers on imbalanced data for software defect prediction problems. SN Comput. Sci. **1**(2), 1–16 (2020). https://doi.org/10.1007/s42979-020-0119-4