SKU JOURNAL OF ENGINEERING RESEARCH Vol-3 Issue-2
(NATIONAL PEER REVIEWED E-RESEARCH JOURNAL)
(April – June 2026)
E-ISSN 3048-930X

EXPLAINABLE SOFTWARE DEFECTS CLASSIFICATION USING SMOTE AND MACHINE LEARNING WITH RBF

Manvendra Singh Divakar
Assistant Professor
Department of Mechanical Engineering
Shri Krishna University, Chhatarpur (M.P.)

ABSTRACT

Programming disappointment expectations and inclinations have long been considered serious problems by IT professionals and programming subject matter specialists. Conventional approaches require prior knowledge of errors or malfunctioning modules in order to identify programming flaws within an application. Robotic programming deficiency recovery approaches, which make use of AI drawings, let the software to detect and recover from programming problems considerably. This part increases the program's efficiency and reduces errors, expenses, and time. A product shortfall expectation improvement model was proposed using AI techniques, perhaps allowing the program to continue with its intended objective. Additionally, we used a variety of improvement assessment standards, such as exactness, f1-measure, correctness, review, explicitness, and Min Max Improvement, to assess the model's performance. The SVM-RBF (Backing Vector Machine-Spiral Premise Capability) AI expectation model process makes use of the concept of AI. The evaluation method demonstrated the high accuracy rate and efficient use of machine learning computations. Additionally, the suggested expectation model is evaluated against several philosophies using a relative metric. The combined findings demonstrated the superior performance of the SVM-RBF method.

***Key Terms**:*
CNN-FS (Convolutional Neural Network with Feature Scaling), precision, recall, specificity, F1-measure, and accuracy.

1. INTRODUCTION

Software bugs typically have an impact on a program's dependability, functionality, and maintenance costs. Without a doubt, since the majority of deformations are covered, it may be difficult to produce code free of errors, even with rigorous claims. One of the most challenging aspects of building computer programs is creating an item bug estimate model that can rapidly identify and eliminate potentially harmful modules. One of the most important tasks in programming enhancement is predicting programming bugs. By identifying dangerous modules before they are incorporated into computer programs, this is done to improve customer satisfaction and overall program performance. Additionally, early programming issue anticipation increases asset productivity and programming flexibility to various situations.

The use of AI methods to forecast programming errors has been the subject of numerous studies. For example, the direct Auto-Relapse (AR) technique can be

used to estimate the number of damaged modules. Based on historical data on aggregated programming errors, the methodology forecasts future programming errors.

Additionally, the investigation used the Root Mean Square Blunder (RMSE) method to compare and contrast the AR model with the Realized Power Model (POWM). For the appraisal investigation, three datasets were also considered, and the results were outstanding. The study examined the likelihood that abnormalities could be predicted using a variety of AI techniques. prior research that is crucial to comprehending each AI strategy and the most recent developments in the use of AI to code bug prediction.

Brain organizations, sometimes known as fake brain organizations, are a collection of deep learning breakthroughs that are modeled after the activity of neurons in the human mind. They are typically employed to handle complex design acknowledgment problems, and they work wonders when decomposing large amounts of information. They work wonderfully when some aspects are unclear and are excellent at handling nonlinear linkages in information.

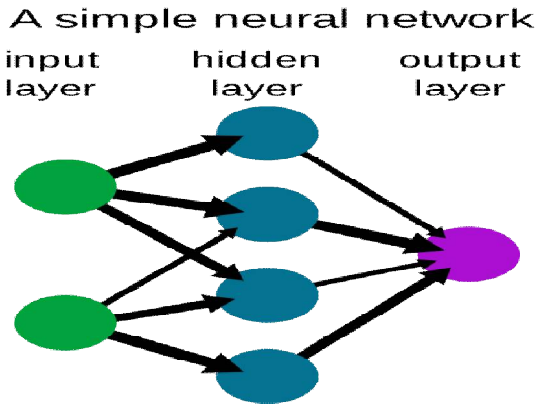


Figure 1: Neural Network

2. LITERATURE REVIEW

In the year 2023, Zhao *et al.* conducted a systematic survey comprising 67 studies on Just-in-Time Software Defect Prediction (JIT-SDP). The survey aimed to advance research and familiarize practitioners with recent progress in JIT-SDP, a variant focused on predicting defects in incremental software changes. Results summarized best practices across JIT-SDP workflow phases, performed meta-analysis of prior studies, and suggested future research directions. Findings indicated that predictive performance correlated with change defect ratio, highlighting JIT-SDP's effectiveness in projects with higher defect ratios. Future directions included domain-specific application, reliability-aware, and user-centered approaches for JIT-SDP.

In the year 2022, Khan *et al.* conducted a systematic literature review on software defect prediction using Artificial Neural Networks (ANN's). The study, which covered publications from 2015 to 2018, aimed to analyze recent trends and critical aspects of using ANN's in defect prediction. The research highlighted the increasing demand for high-quality and cost-effective software systems and emphasized the significance of defect prediction in the software development life cycle. The review was based on publications from IEEE, Elsevier, and Springer, identified eight of the most relevant research works for in-depth analysis, providing valuable insights for researchers

(Xiao-Yuan Jing, Baowen Xu, Haowen

Chen, Yuming Zhou Bing Li; 2022) Heterogeneous deformity forecasting, or HDP, is an imperfection expectation that uses different measures across projects. Because they map source and target data into a shared dimension space where each component has no real significance, the most recent HDP techniques are difficult to understand. Furthermore, HDP categorically opposes class inequality. By reducing the heterogeneity of source and target information and addressing lopsided information while maintaining the true significance of each component of the generated normal measurement space, we intend to provide an exceptional HDP strategy that may address the shortcomings of current HDP approaches.

Table 1. Contributions and limitations of different studies in the literature

Ref. Contributions	Limitations
A data mining approach for predicting defects using static code attributes was explored; they showed promising results in accurately identifying defect-prone modules.	This study was limited to only small-scale systems. It was not exposed to larger and more complex software.
The author introduced support vector machine (SVM) for defect prediction purposes in an open-source software perform a great effect in accurately classifying defective and nondefective modules.	However, the study did not consider the impact of software metrics selection, which could potentially affect the prediction performance
In this state-of-the-art an investigation was conducted to check relationship between code churn and software defects.	There was lack of software metrics, thereby limiting the comprehension of the predictive model.
proposed an ensemble approach for software defect prediction combining multiple classifiers to improve prediction accuracy, approach achieved better results compared to individual classifiers.	The study did not thoroughly analyze the impact of different ensemble configurations.
By concept of transfer learning, proposed a defect prediction model that leverages knowledge from related projects to improve prediction performance.	The study did not thoroughly investigate the transferability in diverse software projects.
In this study, a hybrid model was explored to combine learning and rule-based approaches for defect	The model's limitations, the performance was heavily relied on the quality and effectiveness of the predefined rules.

prediction. While their study showcased the benefits of combining different techniques.	
Proposed a deep-learning-based approach for defect prediction using code change history and static code attributes	The model's performance may be affected as a result the kind of quality of historical data.
The author explores software metrics extracted from both source code and issue-tracking systems for defect prediction. The findings revealed that integrating metrics from multiple sources improved the prediction and accuracy.	The study did not investigate the impact of different weighting schemes for combining the metrics.
Through a comprehensive analysis with non-transformation, time series forecasting method and conventional SGRMs, it has been shown that linear regression with Box-Cox (L_Box- Cox_T) could work well to predict the software fault in short time prediction.	The limitation encountered from the study is SGRM's can only be efficient when there is an increase in the input testing days for software fault prediction.
The authors applied explainable AI techniques to analyze the machine learning models.	The limitations of the paper are they did not evaluate the results with explainable AI extracted features.

3. PROBLEM IDENTIFICATION

Layered neural networks, or ANNs for short, are used in Deep Learning, a branch of Machine Learning, to solve problems. First, the degree to which these predictions differ from the actual results is calculated using a simple loss function. The gradients of this function with respect to the model's parameters are then computed using back-propagation, and the parameters are subsequently updated using an optimizer.

The following list contains the concerns that have been identified based on previous examinations:

- It is usually impossible to identify serious programming flaws.
- The recovery of a product bug is not fully perceived.
- The anonymous programming problem may be identified due to its regrettable correctness.

4. RESEARCH OBJECTIVES

The first step in resolving the problem is identifying the relevant components required for bug prediction. The elements that are prioritized are considered when creating the analytical data collection. Baseline and benchmarking models are implemented

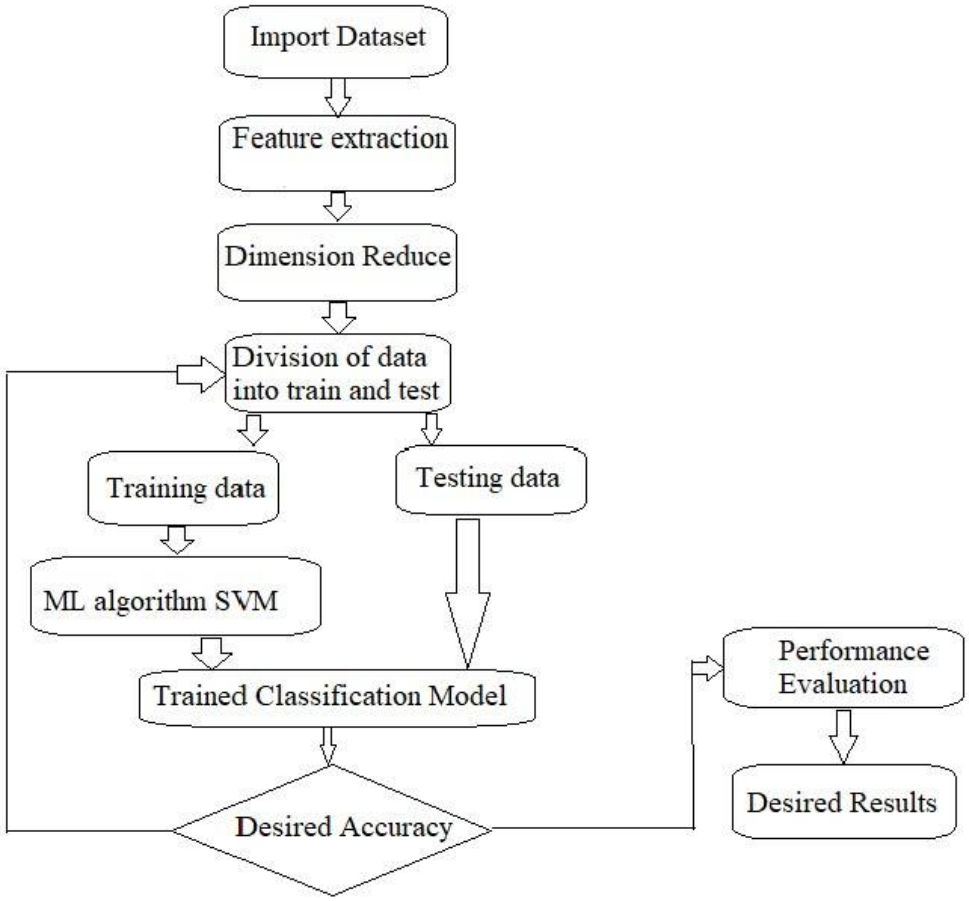
using analytical datasets. Every algorithm is assessed using the k fold validation procedure to ascertain the right accuracy. The following steps must be methodically followed by any supervised machine learning system. These are general frameworks that could help describe the problem more precisely and carry out each of the steps listed below in a systematic manner.

Therefore, the following are the objectives of the proposed work:

a) Throughout the recovery interaction, incrementally check for completely relevant code errors
b) Increase the precision of the programming bug recognition proof; and
c) Increase precision for the best possible recovery of relevant programming errors.

5. METHODOLOGY

Figure 2: Proposed Model of SVM-RBF (Proposed Methodology)



Imagine we have a dataset where each software module is characterized by two features:

X_1: Lines of Code (LOC)
X_2: Cyclomatic Complexity (CC)
The target variable Y is binary:
Y = 1: Indicates that the module contains a bug.
Y = 0: Indicates that the module does not contain a bug.

Table 2: Different Parameters of Data Point

Module	X_1 (LOC)	X_2 (CC)	Y (Bug)
A	50	3	0
B	60	6	0
C	80	5	1
D	90	8	1

6. RESULTS AND DISCUSSION

In addition to identifying both faulty and non-faulty classes, this work focused on automating software fault recovery using a prediction model. In particular, malfunctioning modules are significantly more important than working ones. The effectiveness of software bug prediction was examined in our experiment using the Support Vector Machine with Radial Basis Function (SVM-RBF) method. The various data pretreatment techniques used so far to set the parameters for the software fault model have improved the accuracy and reliability of the classification model.

The following measurements are taken using jupyter notebook and python 3.11.1 on anaconda navigator. Precision, recall, F1-Score, and accuracy are computed as follows when the suggested approach SVM-RBF is applied to CS1.csv from the (PROMISE Dataset):

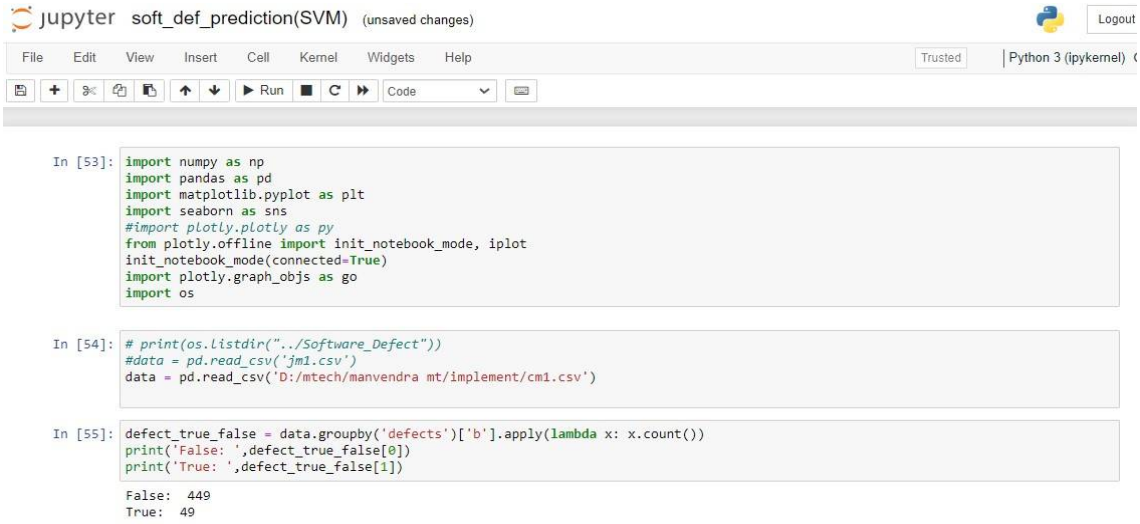
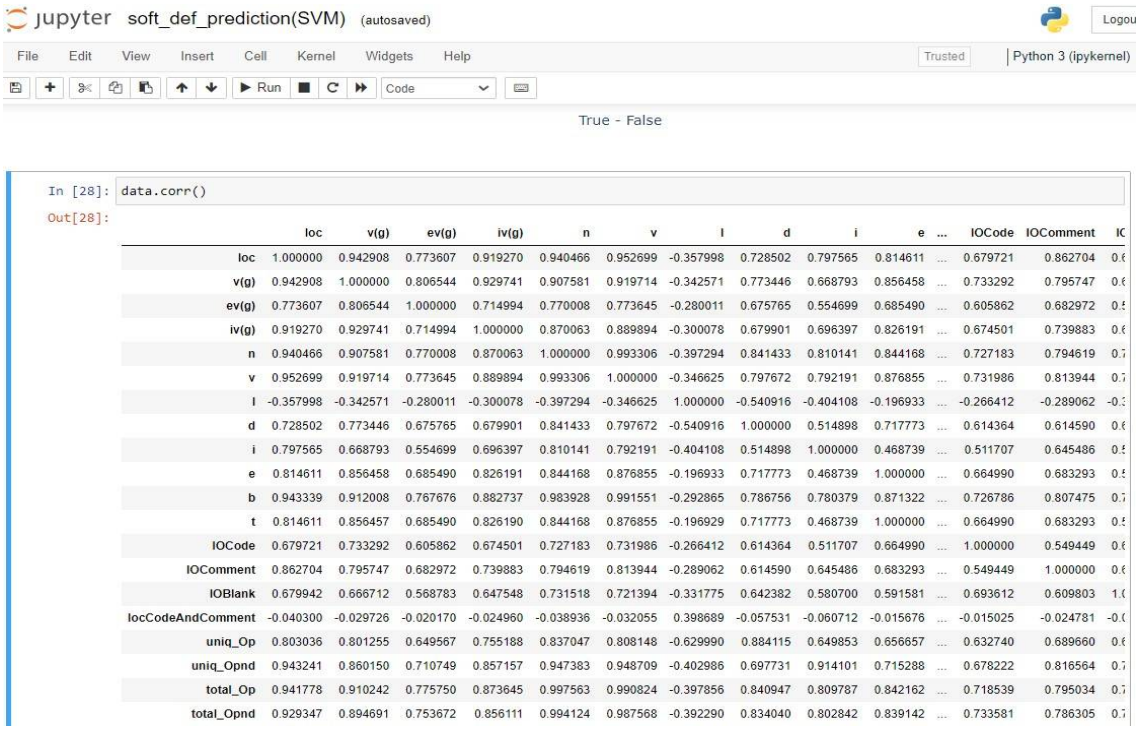


Figure 3: Load dataset and group by content of software defect for SVM-RBF (Proposed Prediction Model)

SKU JOURNAL OF ENGINEERING RESEARCH Vol-3 Issue-2
(NATIONAL PEER REVIEWED E-RESEARCH JOURNAL)
(April – June 2026)
E-ISSN 3048-930X



	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lO
loc	1.000000	0.942908	0.773607	0.919270	0.940466	0.952699	-0.357998	0.728502	0.797565	0.814611	...	0.679721	0.862704	0.6
v(g)	0.942908	1.000000	0.806544	0.929741	0.907581	0.919714	-0.342571	0.773446	0.668793	0.856458	...	0.733292	0.795747	0.6
ev(g)	0.773607	0.806544	1.000000	0.714994	0.770008	0.773645	-0.280011	0.675765	0.554699	0.685490	...	0.605862	0.682972	0.5
iv(g)	0.919270	0.929741	0.714994	1.000000	0.870063	0.889894	-0.300078	0.679901	0.696397	0.826191	...	0.674501	0.739883	0.6
n	0.940466	0.907581	0.770008	0.870063	1.000000	0.993306	-0.397294	0.841433	0.810141	0.844168	...	0.727183	0.794619	0.7
v	0.952699	0.919714	0.773645	0.889894	0.993306	1.000000	-0.346625	0.797672	0.792191	0.876855	...	0.731986	0.813944	0.7
l	-0.357998	-0.342571	-0.280011	-0.300078	-0.397294	-0.346625	1.000000	-0.540916	-0.404108	-0.196933	...	-0.266412	-0.289062	-0.3
d	0.728502	0.773446	0.675765	0.679901	0.841433	0.797672	-0.540916	1.000000	0.514898	0.717773	...	0.614364	0.614590	0.6
i	0.797565	0.668793	0.554699	0.696397	0.810141	0.792191	-0.404108	0.514898	1.000000	0.468739	...	0.511707	0.645486	0.5
e	0.814611	0.856458	0.685490	0.826191	0.844168	0.876855	-0.196933	0.717773	0.468739	1.000000	...	0.664990	0.683293	0.5
b	0.943339	0.912008	0.767676	0.882737	0.983928	0.991551	-0.292865	0.786756	0.780379	0.871322	...	0.726786	0.807475	0.7
t	0.814611	0.856457	0.685490	0.826190	0.844168	0.876855	-0.196929	0.717773	0.468739	1.000000	...	0.664990	0.683293	0.5
lOCode	0.679721	0.733292	0.605862	0.674501	0.727183	0.731986	-0.266412	0.614364	0.511707	0.664990	...	1.000000	0.549449	0.6
lOComment	0.862704	0.795747	0.682972	0.739883	0.794619	0.813944	-0.289062	0.614590	0.645486	0.683293	...	0.549449	1.000000	0.6
lOBlank	0.679942	0.666712	0.568783	0.647548	0.731518	0.721394	-0.331775	0.642382	0.580700	0.591581	...	0.693612	0.609803	1.0
locCodeAndComment	-0.040300	-0.029726	-0.020170	-0.024960	-0.038936	-0.032055	0.398689	-0.057531	-0.060712	-0.015676	...	-0.015025	-0.024781	-0.0
uniq_Op	0.803036	0.801255	0.649567	0.755188	0.837047	0.808148	-0.629990	0.884115	0.649853	0.656657	...	0.632740	0.689660	0.6
uniq_Opnd	0.943241	0.860150	0.710749	0.857157	0.947383	0.948709	-0.402986	0.697731	0.914101	0.715288	...	0.678222	0.816564	0.7
total_Op	0.941778	0.910242	0.775750	0.873645	0.997563	0.990824	-0.397856	0.840947	0.809787	0.842162	...	0.718539	0.795034	0.7
total_Opnd	0.929347	0.894691	0.753672	0.856111	0.994124	0.987568	-0.392290	0.834040	0.802842	0.839142	...	0.733581	0.786305	0.7

Figure 4: Data Correlation in software defect dataset for SVM-RBF (Proposed Prediction Model

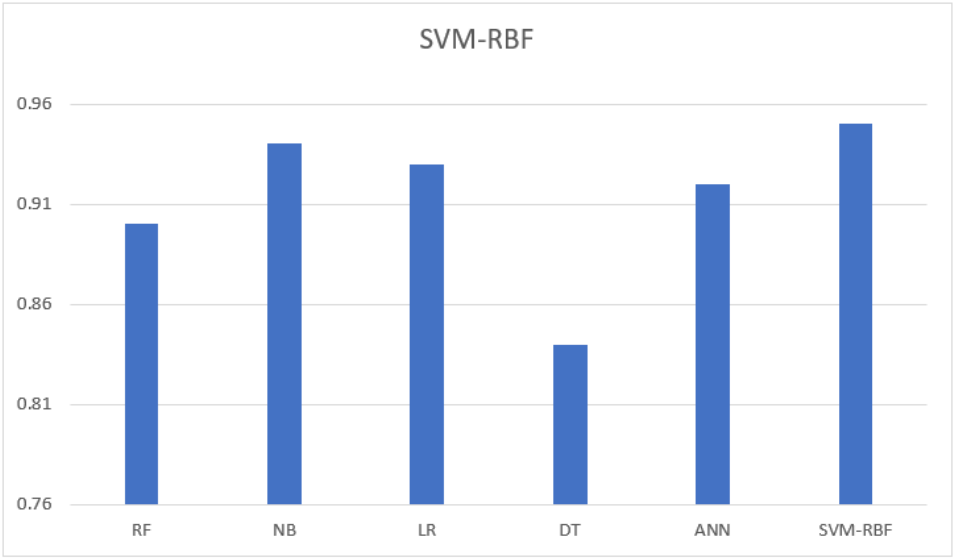


Figure 5: Statistical Comparison of Accuracy between Various Models and SVM-RBF (Proposed Prediction Model)

The accompanying graph illustrates how the proposed approach outperforms earlier

methods in terms of bug prediction recall. The recall of SVM-RBF is 0.12 higher than that of the Decision Tree and the ANN prediction model.

7. CONCLUSION

We have utilized the SVM-RBF model in this appraisal to drop by the typical outcomes. Our examination shows that earlier endeavors didn't give sufficient idea to highlight affirmation and cross underwriting. The proposed system beats different ones concerning accuracy (98.16%) on monster datasets. The blend of the five made approaches yields the best outcomes since it is computationally referencing (it keeps away from overfitting and gives quick suspicion speeds) and versatile in application (it very well may be utilized for both apostatize and depiction issues). Taking a gander at this procedure further for bug supposition in huge learning models has been a consistent undertaking.

There ought to be terminations to these:

1.In terms of precision, the proposed model performs better than SVM-RBF.The precision is 0.01 higher than that of Naïve Bayes.

2. The proposed model performs better in terms of recall than SVM-RBF Regression. The recall of SVM-RBF is 0.12 higher than that of the Decision Tree and the ANN prediction model.

3. In terms of F1-Score, the suggested model performs better than the SVM-RBF. Compared to Random Forest and ANN, there is a 0.08 increase.

4. The proposed model is more accurate than ANN. The accuracy has increased by 13.23%.

SUGGESTIONS FOR FUTURE WORK

Our suggested strategy improves diagnostic precision, which is crucial for effective therapy. Future improvements should include testing the accuracy with new datasets and using other AI techniques to double-check the estimation's precision. Due to the massive quantity of data needed for performance estimation of train data, the proposed model has a processing time restriction. In the future, the same algorithms will be used to data in real time in order to estimate the system's efficacy.

REFERENCES

1. Görkem Giray, Kwabena Ebo Bennin, Ömer Köksal, Önder Babur, Bedir Tekinerdogan, “On the use of deep learning in software defect prediction”, The Journal of Systems & Software, 2023.
2. Iqra Batool, Tamim Ahmed Khan, “Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review”, Computers and Electrical Engineering, May 2022.
3. Haowen Chen, Xiao-Yuan Jing, Yuming Zhou, Bing Li, Baowen Xu, “Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction”, Information and Software Technology, July 2022.

4. Cagatay Catal, Görkem Giray, Bedir Tekinerdogan, Sandeep Kumar & Suyash Shukla, "Applications of deep learning for phishing detection: a systematic literature review", Knowledge and Information Systems , 2022.
5. Xieling Chen, Haoran Xie, Zongxi Li, Gary Cheng, "Topic analysis and development in knowledge graph research: A bibliometric review on three decades", Neurocomputing, October, 2021.
6. Cagatay Catal, Görkem Giray, Bedir Tekinerdogan, "Applications of deep learning for mobile malware detection: A systematic literature review", 2021.
7. Konstantin S. Kobylkin, Anton V. Konygin Ilya P. Mezentsev and Vladimir E. Misilov, "A Survey on Software Defect Prediction Using Deep Learning", IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2021.
8. Farah Atif, Manuel Rodriguez, Luiz J. P. Araújo, UtihAmartiwi, Barakat J. Akinsanya & Manuel Mazzara "A Survey on Data Science Techniques for Predicting Software Defects", IEEE Conf. of Software Engineering, 2021.
9. Saleema Amershi; Andrew Begel; Christian Bird; Robert DeLine; Harald Gall; Ece Kamar; Nachiappan Nagappan, "Software Engineering for Machine Learning: A Case Study", IEEE Conf. on Machine Learning, 2019.
10. [10]George G. Cabral; Leandro L. Minku; Emad Shihab; Suhaib Mujahid, "Class Imbalance Evolution and Verification Latency in Just-in-Time Software Defect Prediction", IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019.
11. Kwabena Ebo Bennin; Jacky Keung; Passakorn Phannachitta; Akito Monden; Solomon Mensah, "MAHAKIL: Diversity Based Oversampling Approach to Alleviate the Class Imbalance Issue in Software Defect Prediction", IEEE Transactions on Software Engineering, 2018.
12. Yuxiang Gao, Yi Zhu, Yu Zhao, "Dealing with imbalanced data for interpretable defect prediction", IEEE Conf on Data Analysis, 2017.
13. Kwabena Ebo Bennin; Jacky Keung; Akito Monden; Yasutaka Kamei; Naoyasu Ubayashi, "Investigating the Effects of Balanced Training and Testing Datasets on Effort-Aware Fault Prediction Models", IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016.
14. Faruk Arar, Kürşat Ayan, "Software defect prediction using cost-sensitive neural network", Applied Soft Computing, Volume 33, August 2015.
15. Deepika Badampudi, Claes Wohlin, Kai Petersen, "Experiences from using snowballing and database searches in systematic literature studies", 19th International Conference on Evaluation and Assessment in Software Engineering, 2015.
16. Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", Computation and Language, 2014.
17. N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, "SMOTE: Synthetic

Minority Over-sampling Technique", Artificial Intelligence, 2011

18. Gul Calikli; Ayse Tosun; Ayse Bener; Melih Celik, "The effect of granularity level on software defect prediction", 24th International Symposium on Computer and Information Sciences, 2009.
19. Cagatay Catal, Banu Diri, "A systematic review of software fault prediction studies", Expert Systems with Applications, Volume 36, Issue 4, May 2009.
20. Alessandro Birolini, "Reliability and Availability of Repairable Systems", Reliability Engineering, 2004.