International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

A Review of Automated Software Bug Severity Prediction from Traditional to Advanced Models

Sweety Ahlawat[1], Dr. Dhiraj Khurana[2]

Submitted:01/02/2024 **Revised:** 14/03/2024 **Accepted:** 24/03/2024

Abstract: Predicting the severity of bugs is an important part of software quality assurance throughout both the development and maintenance phases, so developers may better utilize their resources. An exhaustive evaluation of software defect severity assessment models using machine learning, deep learning, and statistical techniques is presented in this work. In the context of software maintenance, severity refers to the efficacy, resistance, and hindrance of bugs. Additionally, this study delves into the methods used for prediction and various bug attributes. It discusses popular data repositories, pipeline workflows, severity techniques, and current research directions based on the existing system. Evaluate several feature sets based on their performance in predicting problem severity levels, including model complexity, generalizability, and data availability.

Keywords: *Bug Severity, Machine Learning, Deep Learning, Bug Features, Severity Prediction.*

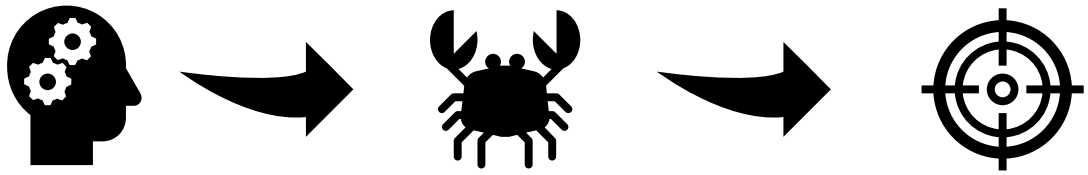
1. Introduction

Software performance can be severely impacted by bugs, faults, and flaws; in certain cases, this can even result in the software becoming unusable. Over the time, there has been an exponential increase in the use of software due to necessity for automated and effective system. With little to no human intervention, embedded software has decreased errors and made accelerated processes in digital and electronics systems. It is impossible to reap the benefits of precise and efficient software without the use of software bug detectors. Software detectors check bug severity and priority. Software bug severity [1] –[6]classify in levels. These levels can be low, medium, high and critical. Priority of a bug [7] – [15] for addressing and assigning to developers. When people manually assign severity ratings to bug reports, it can lead to biases and mistakes in the process. Consequently, methods for automatically forecasting severity levels are attracting more and more attention.

A number of methods make use of data retrieved from bug reports, such as the reporter's attributes, information on the impacted product and its parts, and textual data retrieved from the summary and description fields [16] - [23] Deep learning algorithms have received less attention than other methods for predicting severity based on a variety of criteria, such as text content analysis and engineering considerations. Using the complex, multi-dimensional structure of bug reports, deep learning might enhance the accuracy of severity estimates. More research and development is needed in this sector since its full potential has not yet been realized in this context. As part of the routine upkeep, software triagers search for and grade the severity of bugs.

[1] *Department of Computer Science and Engineering, Maharshi Dayanand University, Rohtak-124001, India. Email - sweetyahlawat6@gmail.com*

[2] *Department of Computer Science and Engineering, Maharshi Dayanand University, Rohtak-124001, India. Email - dhirajkhurana@mdurohtak.ac.in*



Software Developer Software Bug Check Level of Severity

Goal of this paper is as follows:

1. Reviewing past methodologies based on machine and deep learning to predict bug severity

2. Examining the gap in research and delving into potential avenues for future research.

The other sections of this study are compressed as: Depicts an overview of methodologies used to predict bug severity in Section 2. Section 3 discusses the existing literature. Section 4 elucidates the rationale behind undertaking this research endeavour. Section 5 presents the motivation behind this work. Current research and future direction are discussed in Section 6. Section 7 concludes the work.

2. Methodology

The area of software bugs has experienced a notable expansion in recent times owing to an increased number of researchers who have become intrigued by this particular domain. This study for software bug severity is based on different approaches. Findings from this extensive literature review highlight the importance of machine learning (DL) techniques for assessing software defect severity. Extensive literature on bug severity was conducted for ML, DL, NLP, synthetic minority oversampling technique and many others from various scientific databases – 'Springer', 'IEEE', 'Web of Science', 'Scopus', 'Elsevier' and 'Science Direct'. This screening yielded 15,230 records, as identified by the literature search. Among these publications, selected papers in software 2,600 and from them, nearly 72 papers specified for bug severity. A small number of publications were examined for this study since its primary objective was to establish standards for the methodology and criteria of machine learning and deep learning research. Roughly seventy-two research and review articles make up this extensive analysis.

Presently, many researchers are working on bug severity in software. The annual breakdown of research papers reviewed for this study is detailed in Figure 1.

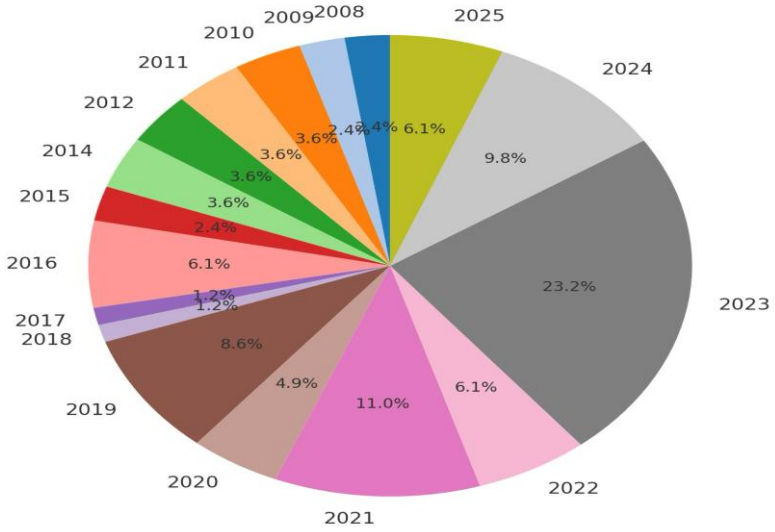


Figure 1: Breakdown of research papers

The distribution of papers over the years demonstrates varying levels of research activity within the specified period. Notably, there is an upward trend in the number of papers published from 2008 onwards, peaking at a high rate of paper in 2023(recent year) with 19 papers. However, this trend is followed in 2024, with eight papers up to February and some of 2025.

Table 1: Comparison of existing studies with the proposed framework

Sr No.	Author (s)	Bug Severity	Bug Prioritization	Bug Categorization
1	Ahlawat, S. & Khurana D. [24]	YES	NO	NO
2	Vaid, K., & Ghose, U. [25]	YES	YES	NO
3	(Panda & Nagwani, [26])	YES	YES	NO
4	(Ali et al. [27])	YES	NO	NO
5	(Liu et al., [28])	YES	NO	NO
6	(Panda & Nagwani, [29])	NO	NO	YES
7	(Hamdy & El-Laithy, [30])	YES	NO	YES
8	(Hamouri et al., [31])	YES	NO	NO
9	(Meesad et al., [32])	YES	NO	NO
10	Shatnawi, M. Q., & Alazzam, B. [33]	YES	YES	NO
11	(Tabassum et al., [34])	NO	NO	YES

The suggested framework is compared to previous studies in Table 1. As a result, we can see that the proposed framework sort bugs according to priority, categorization, and severity. Several writers discuss the importance of prioritizing and severity of bugs in their work, including Vaid and Ghose [25], Panda and Nagwani [29], and Shatnawi and Alazzam [33]. Bug categorization, however, needs further attention as it has been the subject of so few studies (e.g., Panda and Nagwani [26] and Hamdy and El-Laithy [30]. While some studies comprehensively cover multiple dimensions of bug management, others focus on specific areas, highlighting the diverse approaches employed in the field.

3. Related Work

Many text-based predictions methods have been proposed for determining the severity of bug reports when machine learning technology is used to look at software artifacts. (Rao et al. [35]) adopted PCA for choice of feature s and SMOTE to address class imbalance. They assessed the degree of code smell using five machine learning techniques, and they were very ac curate, especially when using the Random Forest a nd Decision Tree methods.The D.C. and G.C. datas ets performed well in terms of accuracy, precision, recall, and Fmeasure, and the Random Forest mode l performed very well, obtaining up to 0.99 accurac y for Long method code smells, according to the st udy.

(Dewangan et al. [36]) proposed code odour prediction using machine learning algorithms. Four code smell datasets—God-class, Data-class, Feature-envy, and Long-method—generated from 74 open-source systems are considered. Model performance was assessed using a 10-fold cross-validation methodology using code smell datasets. Prediction accuracy was improved by using Chi-squared and Wrapper-based feature selection techniques. Grid search optimization enhanced algorithm performance; Logistic Regression achieved 100% accuracy for the Long-method dataset, while Naive Bayes performed at 95.20%,

the lowest of all algorithms using Chi-squared feature selection.

(Pandey & Tripathi, [37]) addressed the effects of noise and the Class Imbalance problem on five baseline Software Defect Prediction (SDP) models. They manually introduced noise levels (0 to 80%) and determined how they affected the models' performance. Additionally, we have proposed the SDP model, which outperforms other traditional approaches and has the maximum noise-tolerating ability. When 10% to 40% more noisy examples are added, the baseline models' True Positive Rate (TPR) and False Positive Rate (FPR) values drop by 20% to 30%. Similarly, SDP models' ROC (Receiver et al.) values drop from 40% to 50%. Comparably, SDP models' ROC (Receiver et al.) values drop from 40% to 50%. Compared to other conventional models, the proposed approach results in a 40% to 60% reduction in noise. They performed 864 experiments on three public datasets. After comparing different machine-learning techniques, they examined that the most minor deviation by R.F., whereas SVM and AdaBoost have high variation toward the noise.

(Fan et al. [38]) proposed ML algorithms and selected 33 features for software bug prediction from bug reports. The five dimensions—reporter experience, cooperation network, completeness, readability, and text—were used to organize the derived features. They employed a RF classifier based on these variables to determine which bug reports were legitimate. Using large datasets of 560,697 bug reports from five open-source projects, the researchers validated their methodology. F1-score 0.74 of authentic bug reports and from all datasets achieved AUC of 0.81.

In their study, (Pecorelli et al.[39]) discussed about prioritization by employed machine-learning models to classify severity level by the developers. F- measure score up to 85 % and comparison between different approaches. (Huang et al. [40]) evaluated Explainable Artificial Intelligence (XAI) based three to five profound attributes. They increased in accuracy complexity with 5% to 29 % yield results. Its findings additionally demonstrate the necessity it is to distinguish between coarse-grained forecasting intends and create fine-grained feature engineering methods.

(Dewangan et al. [41]) presented merged of three machine-learning approaches. They utilized Principal Component Analysis (PCA) for reduction of dimensions and for successfully navigating datasets by choosing different components from dataset. 10-fold cross-validation method employed better performance of models. In result, LR model notable result is maximum with 99.97% on data class dataset. Contrary from present LR, KNN gives least precision of 77.38 % revealing a variability in accuracy of the model across multiple elements within it

(L. Rehman [42]) examined with Machine-Learning techniques on open-source datasets from six repositories. To anticipate persistent issues experts deployed collaborative methods leveraging best heuristics by employed DT, RF, SVM, Multi-Layered Neural Networks and KNN for bug prediction. Neural networks obtained evaluation scores of 0.985 for accuracy, 0.98 for precision, recall, and F1-Measure on the GCC dataset. (Peralta et al. [43]) Analyzed bug report qualities using Bayesian Network (B.N.). Two repositories are used to evaluate bug reports: Bugzilla and Mozilla. The performance of the classifiers is based on resolution and fixing time for training and testing the samples. Performance-based testing of B.N. in fixing time and resolution were 0.927, 0.94, 0.975 and 0.744, 0.75, 0.956 accuracies, recall and precision, respectively.

(Chmielowski et al. [44]) proposed a novel method for software bug reports using machine-learning models and human factor-based techniques. They chose data from November 2021 to January 2022 to make decisions about the transfer, which was 79% and 66% resolved for suggestions. In machine learning, decision accuracy in November and December for resolved was 38%, and for not fixed type of resolution, 55%. In December, the resolution was 50%, and the fixed type of resolution was 80%. At the same time, human accuracy was better than the ML model. Both machines and humans needed to be corrected in December 28% to resolve and fix the issue.

(Nagwani & Suri, [45]) examined the six categories of software bug-triaging approaches (SBTT). They examined that in 2005, Machine learning and tossing Graph techniques were used; in 2010, Topic Modeling, Information Retrieval, and Fuzzy sets; and in 2015, Social Network Analysis; and in 2020, Deep Learning, Recommender systems, and Document embedding were employed. Datasets choose from Open Office, Net Beans, Mozilla, and

Eclipse repositories. They carried out (PRISMA) and selected 123 studies for Artificial Intelligence (AI) based review.

(Agrawal and Goyal [46] explored Word2Vec's impact on bug severity prediction and assessed different averaging methods and hyperparameter settings. Bug tracking systems rely on repositories to enhance software quality by managing and categorizing bugs. Manually assessing and assigning bug severity is impractical, necessitating precise classification to prevent mislabelling by non-experts. Text mining, specifically word embedding, can effectively analyze vast databases of textual bug reports. Word embedding, like Word2Vec, encodes words as real-valued vectors, capturing their meanings. However, optimal Word2Vec performance depends on hyperparameter configuration and feature selection, requiring careful tuning. Findings indicate that larger window sizes enhance classifier performance, but the minimum word count parameter's effect varies with the dataset. For classes with limited records or rare class-specific words, Random Forest and Xgboost classifiers perform well, while Support Vector Machine and Naive Bayes perform better. (Zaidi & Lee, [47]) presented a graph representation method that addresses the node classification problem with the bug triage problem for the bug reports dataset. They employed JDT, Eclipse, and two different Firefox throughout the bug dataset. Occurrences of word-to-word are documented in the heterogeneous form of a graph. An open-source project's bug report graph representation was used to train the graph convolution network (GCN). Using Top-K accuracy on bug reports from repositories like JDT, Platform, and Firefox, which calculates accuracy from Top 1 to Top 10, the efficacy of the approach was evaluated.

4. Background

An important procedure that evaluates different bug aspects to establish severity levels, from severe to trivial, is bug severity prediction. It helps prioritize issue fixes, allocate resources optimally, and improve quality control. Additionally, it makes proactive risk management possible by locating and resolving important problems. Statistical analysis and machine learning are two methods used to guarantee the success and dependability of software systems.

(Gujral et al. [48]) recommended a technique which is based on dictionary for classifying severity of bug. If there are a lot of reported defects, the manual procedure of determining the severity of the bugs gets highly complicated. Therefore, it should be automatically classified so that faults that require urgent attention are fixed as soon as possible. Researchers have tried automating the task a few times. Work aims to automate the classification of bug severity using text mining techniques and the 'Naïve Bayes Multinomial classifier'. This paper also suggests employing a vocabulary of bug phrases to improve the efficiency of this task. (Zhang et al. [14]) fixed recommendation for software bugs to predict severity. Firstly, they search for historical reports comparable to a new bug using the 'K-Nearest Neighbor' (KNN) classification and the modified REP method. The outcomes showed that the approach can enhance severity prediction and fixer recommendation performance. The method uses the upgraded version of REP, REP_{topic}, as a similarity metric to retrieve the top-K nearest neighbours of the newly reported problem. The next step is to apply the elements of the K neighbors in order to achieve severity prediction and semi-automatic fixer recommendation. These aspects include the developers who were involved and the language similarities with the bug report that was distributed.

(Kaur & Kaur [49]) deliberated a comparative analysis for software fault prediction. Research attempts to determine the optimal category, classifier—for fault classification. A total of twenty-one classifiers from five different categorizations were applied to evaluate the performance of five open-source programmes using object-oriented metrics. Several classification models have been assessed using MATLAB's classification LearnerApp. The study suggested using SVM and Ensemble methods instead of KNN, Regression, and Tree. Five datasets and four performance metrics have been used in the comparison. These five open-source project reports are (PMD, Find Bugs, Dr Java, EMMA, and Trove) for experiments. (Mehta & Patnaik, [50]) discussed machine learning methods in this research to improve the prediction of software bugs. The suggested model combines 'Partial Least Square' (PLS) 'Regression, Recursive Feature Elimination' (RFE) for dimension reduction, and the Synthetic Minority Oversampling Technique

employed for imbalanced datasets. On datasets from NASA and PROMISE archives, XGBoost in conjunction with the Stacking Ensemble approach produced fault prediction accuracy greater than 0.9. In terms of accuracy, the Multilayer Perceptron (MLP) model fared better than other models, especially when dropout, SMOTE, and correlation-based feature selection were included.

(Ramay et al. [20])analyzed severity prediction based on the Deep Neural Network of bug reports. They enhanced the f-measure by 7.90% on average. To retrieve bug reports from 'Eclipse' and 'Mozilla projects' , they looked into the 'Bugzilla bug repository'. They gathered bug reports but disregarded enhancement and duplicate reports. They choose seven open-source products' bug reports from the dataset - Platform, Thunderbird, Firefox, CDT, JDT, Core, and Bugzilla. There have been 59616 bug reports in total, of which 8.39%, 16.77%, 16.77%, 16.77%, 16.77%, 16.77%, 16.77%, and 7.76% are related to each product, in that order. (Sharma et al. [51]) presented deep learning models can identify scents without requiring a lot of feature engineering. Second, look into the potential use of transfer learning to identify code smells. Next, use the primary hidden layers of convolution and recurrent neural networks to train models for smell detection. First, use C# code samples to train and evaluate the models; second, use Java code examples to evaluate the models and train the models from C# code, and vice versa. According to the investigation, the deep learning models' performance varies depending on the smell. They deny the claim that for all odours under consideration, RNN models outperform CNN models.

4.1 Bug Data Representation

Monitoring software bug tools according to bug reports includes several software attributes. The summary and description are the two main text-based characteristics of a software problem.

Table 2: Software Bug attributes with their description

Sr No.	Bug Attributes	Description
1	ID	represents the unique integer number.
2	Title	represents the summary of the reported bug in one/ two lines.
3	Priority	refers to the impact or seriousness of a bug on the software system
4	Severity	determines the order in which bugs should be addressed based on their relative importance or urgency
5	Component	Component of Software
6	Bug State	Defines the present state of the bug
7	Modified	The most recent date of the software for updating
8	Submitted	Date of time for submission
9	Assign to	Current bug is assigned to the developer
10	Bug Reported by	Software bug identified and reported by the person
11	Description	Comprehensive details regarding the reported bug
12	Keywords	Bug keywords are specific terms or phrases used to describe a bug's nature, behaviour, or characteristics in software development.
13	Resolution	For process of resolving and rectifying reported bugs or issues in software development

Table 2 lists several bug characteristics. In addition to bug priority, severity, affected component, status, modification date, submission date, assigned developer, reporter, and bug description, bug reports include distinct identifiers such as bug I.D. and bug title. These characteristics work together to

give a thorough picture of every reported bug, which helps manage and fix it during software development.

4.2 Data Accumulation and Benchmarked Repositories

Bug data is collected and retained up to date in bug tracking or problem-solving tracking systems during the software development process. Today, there are an extensive number of standard bug-tracking systems that are commonly used in software development. Testing specialists find errors in the program and file bugs in the software. After that, the effectiveness of the bug should be checked to see how much software influences the bug.

The academic assessment that is presented states that most study was done with open-source bug repositories like as Mozilla (Gomes et al. [52]), Panda at al. [53], Kukkar et al. [54] Olaleye et al. [55], Hazra et al. [56]), Eclipse (Shatnawi et al. [33], Kim et al. [57], Bibyan et al. [58], NASA (Bibyan et al. [58], Nabil Mahmoud et al. [59]; Mamatha et al. [60], Apache (Panda and Nagwani [26], Olaleye et al. [55]) etc. along with these repositories Netbeans also used by (Panda and Nagwani [26]) and many others. Table 3 lists the popular repositories with their corresponding URLs.

Table 3: Software Bug Repository with its URL

Sr.No.	Repository	URL
1	Eclipse bug reporting portal	'https://bugs.eclipse.org/bugs/'
2	NetBeans error tracking system	'https://apache.org/jira/projects/NETBEANS'
3	OpenOffice defect service	'https://wiki.openoffice.org/wiki/Issue_Tracker'
4	NASA bug repository	'https://heasarc.gsfc.nasa.gov/ftools/users/node19.html'
5	Bugzilla bug repository	'https://bugzilla.mozilla.org/'
6	WineHQ bug repository	'https://bugwinehq.org/'
7	TensorFlow bug repository	'https://github.com/tensorflow/tensorflow/'
8	GCC issue tracking network	'https://gcc.gnu.org/bugzilla/'
9	GitHub defect database	'https://github.com/orgs/github/repositories'
10	Apache error reporting system	'https://httpd.apache.org/bug_report.html'

Researchers typically use the main bug repositories and their accompanied URLs, which are clearly outlined in Table 3. Some different type of datasets employed to predict bug severity by number of researcher in their research which is shown in Figure 2.

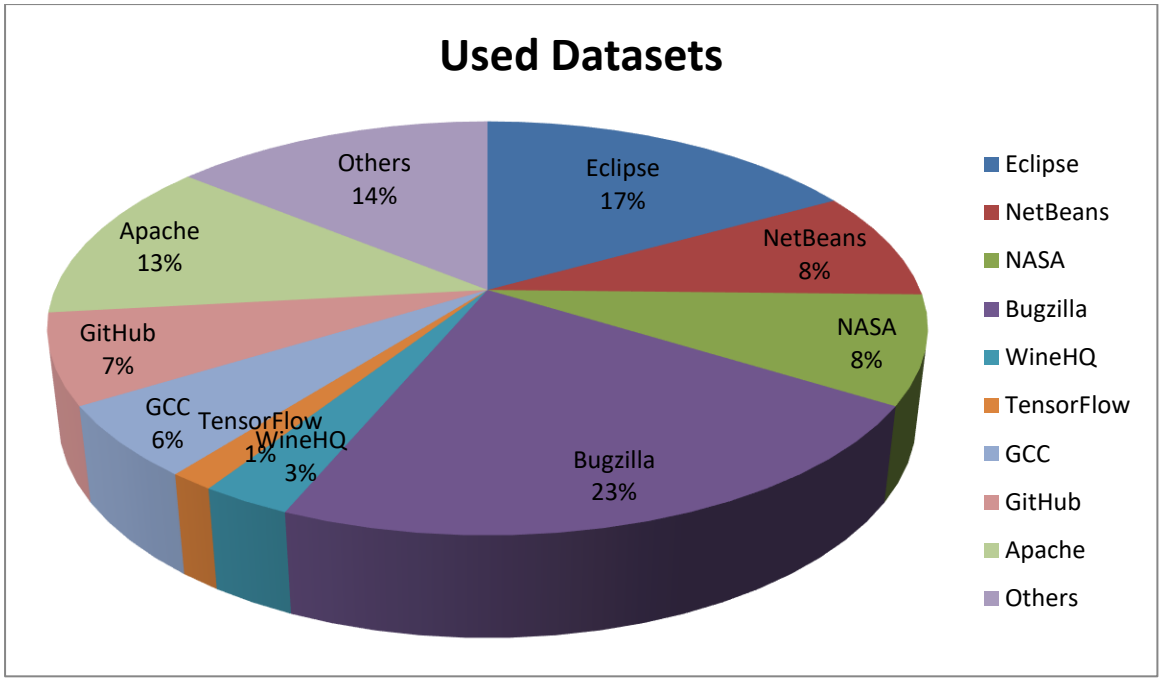


Figure 2: Distribution of used Datasets

4.3 Bug Severity Pipeline

The bug severity pipeline shown in Figure 3 encompasses a systematic process for assessing, prioritizing, and resolving reported bugs in software development. Beginning with bug reporting, the pipeline proceeds through bug triage, wherein bugs are reviewed and categorized based on severity, priority, and impact.

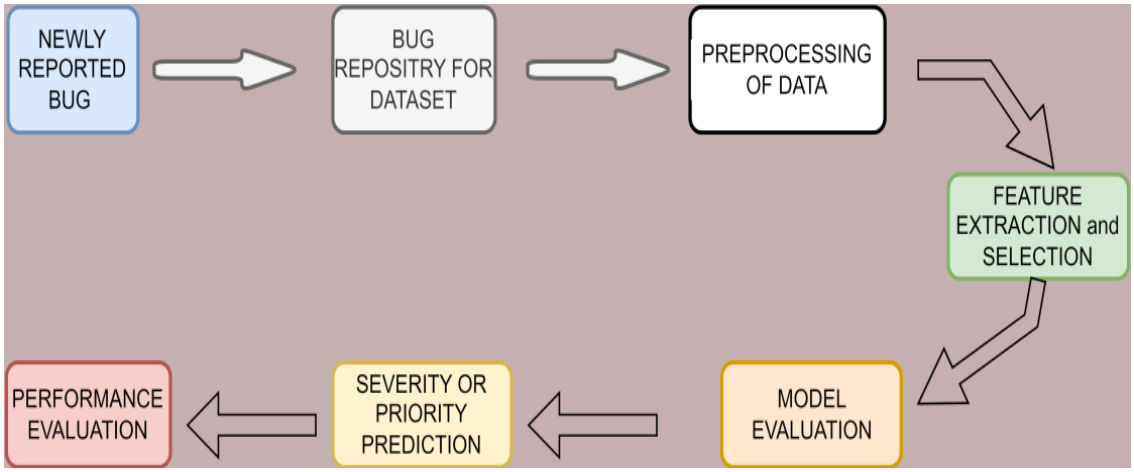


Figure 3: Workflow diagram of Software Bug Severity Process

Bugs are fixed, tested, validated, prioritized, their severity evaluated, and closed as part of the process. Firstly, reported a new bug from dataset and then start extraction and selection of features after then select model, by employing selected model predict the severity of a bug, at last evaluation of result will be done. By doing this, bug-fixing efforts are effectively managed and prioritized, preserving the dependability and quality of the bug.

4.4 Bug Severity Analysis of Methods

Software defects severity is analysed and predicted using sophisticated approaches such as machine learning, Natural Language Processing (NLP), optimization algorithm They support developers in enhancing user experience, software quality, and resolution efforts prioritization. According to literature review, this research explored some techniques in Figure 4.

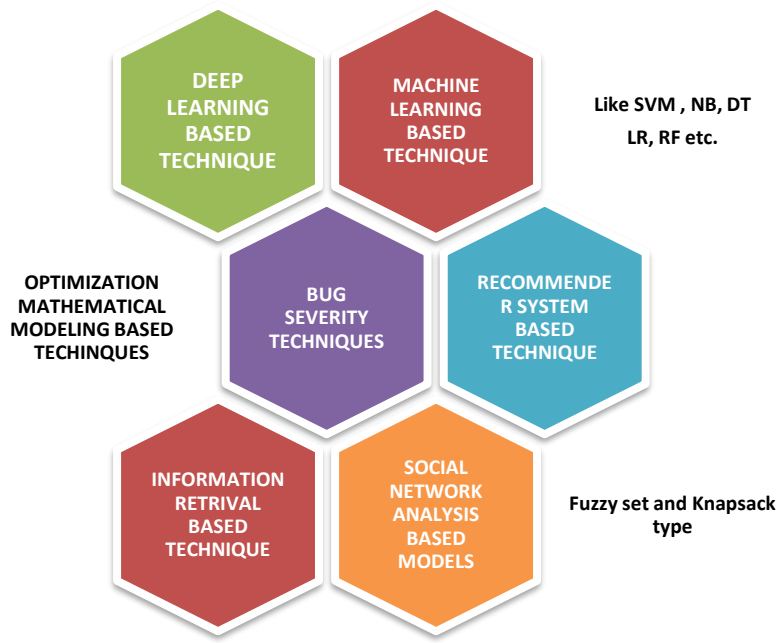


Figure 4: Bug Severity Techniques

Figure 4 demonstrates the ways we spoke about before for checking how well the bug works. There is a distinct part of the graphic that shows this approach and how it fits into the whole procedure. The arrows that link these strategies show how they are related and work together to make bug severity analysis and prediction easier. This visual representation elucidates the diverse array of strategies employed in addressing and mitigating software bugs, in an effort to improve the overall quality of software and make bug fixes more straightforward. Table 4 outlines the characteristics of each 'Bug Severity Technique'.

Table 4: Characteristics of different Bug-Severity Techniques

Sr. No.	Bug Severity Techniques	Characteristics
1	Machine Learning (ML)	Train data and categorize
2	Latent Dirichlet Allocation (LDA)	Statistical Model
3	Synthetic Minority Oversampling Technique(SMOTE)	Encoding of characteristics
4	Deep Learning (DL)	Stack for acquiring and categorizing data in layers
5	Recommender System (R.S.)	Recommend according to user choice
6	('Bidirectional Encoder Representations from Transformers')BERT	Estimates the impact of defect reporting on mobile app maintenance using a DNN.
7	Fuzzy Similarity Measure-Based Software Bug Severity Prediction (FBSP)	To predict the severity of software bug
8	Ant colony optimization (ACO)	Estimating the severity of defects and grouping them into multiple types based on severity
9	Natural Language Processing (NLP)	Handle the bug report content; n-gram will extract the characteristics

The methodologies mentioned above collectively present diverse strategies for assessing and forecasting bug severity within software systems. They cover a variety of methods, such as novel approaches, machine learning, natural language processing, and techniques for optimization. Their integration into software development processes promises to enhance the quality of software products and the efficiency of bug resolution procedures.

According to the paper's analysis, bug representation is mainly four sections. This research examines the results of most of the authors in vector space node representation. Figure 5 represents the severity result-wise.

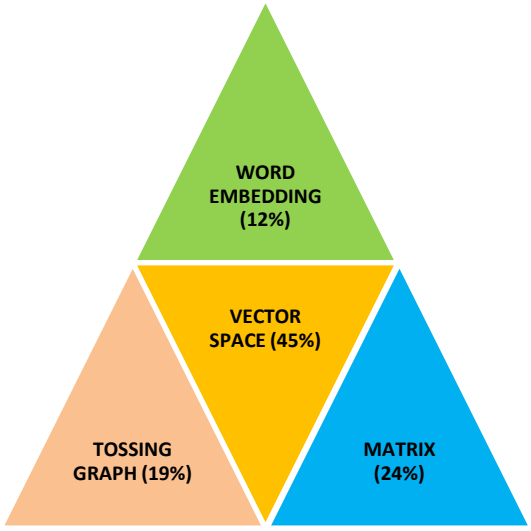


Figure 5: Representation-wise Bug Result

4.5' Machine-Learning & Deep-Learning (ML/DL) Techniques' in bug severity

Predicting bug severity is crucial for software quality assurance and maintenance. Machine learning techniques can be applied to this problem to classify bugs into severity categories based on various features. Various machine-learning techniques are used to predict bug severity. Naïve Bayes has been used in (Nabil Mahmoud et al. [59], Singh & Chhabra [61], Mamatha et al. [60], Gomes et al. [52] , Kukkar et al. [18], Hamouri et al. [31], Shatnawi et al. [33], Bibyan et al.[58], Johnson et al. [62])to predict bug severity. 'Support vector machine' has been used by (Gomes et al. [52], Vaid et al. [25] , Kukkar et al. [54] , Hamouri et al. [31], Meesad et al.[32], Hao et al. [63], Shatnawi et al. [33] to predict bug severity. 'Random Forest' (Gomes et al. [52] , Meesad et al. [32], Shatnawi et al. [33], Bibyan et al. [58]). 'Convolution Neural Network' has been used in (Yoo et al. [64] , Hao et al. [63], Abdou and Darwish [65] for severity prediction.

5. Motivation

The significance of bug severity in software development and quality assurance is highlighted in this literature review. Accurately predicting defects is crucial in today's complex software world. Researchers look at different methods, tactics, and algorithms for estimating bug severity, taking note of common challenges and limitations. The goal of the review is to provide more trustworthy prediction models and ways for finding and fixing problems early on in software, which would enhance software quality assurance. Predicting and managing the severity of issues in complex systems requires an in-depth understanding of several methods and strategies. Researchers and practitioners do thorough literature reviews to gain a full understanding of the approaches, strategies, and algorithms used for bug severity prediction.

Today's dynamic software environment necessitates precise bug severity prediction and management due to quick development cycles and complicated systems. Investigating various data sources, feature extraction techniques, machine

learning models, and assessment measures from earlier research are all included in this.

6. Current research and directions

Current research in software bug severity to predicate the advance model of various techniques to improve its accuracy. To further improve severity prediction models' adaptability and generalizability across various software projects and domains, there is an increasing interest in adding contextual data, such as project-specific characteristics, team dynamics, and organizational procedures. Various techniques mentioned in section 4. On the basis of these techniques, this research summarized current work in bug severity in Table 5.

Table 5: Bug Severity related to summarized existing study

Sr. No.	Author(s)	Title	Techniques	Published	Classifier	Datasets	Performance
1	(Nabil Mahmoud et al. [59])	"A proposed model for detecting defects in software projects"	Machine Learning	'Indonesian Journal of Electrical Engineering and Computer Science'	D.T., NB, Neural Network, Association Rule	Data from software business and NASA datasets used	D.T.- 75, NB- 72, N.N.- 71, Association rule – 76, Proposed Model – (L.R.) – 85 in terms of accuracy
2	(Panda and Nagwani [26])	"Software bug severity and priority prediction using SMOTE and intuitionistic fuzzy similarity measure"	'Synthetic Minority Oversampling Technique' (SMOTE) and 'Intuitionistic Fuzzy Similarity Measure' (IFSM)	'Applied Soft Computing (Elsevier)'	Severity Prediction (IFSMSP), IFSMPP	Eclipse, Mozilla, Apache, and NetBeans	Eclipse data set, results are in terms of accuracy, precision, recall, and F-measure 0.923, 0.906, 0.919, 0.912 and for Mozilla dataset are 0.930, 0.919, 0.922, 0.920, and For the Apache measured values are 0.895, 0.898, 0.908, 0.903 and For the NetBeans 0.936, 0.922, 0.929, 0.925. Respective

							ly.
3	Vaid, K et al. [25]	"Machine Learning-Based Predictive Analysis of Software Bug Severity and Priority"	Machine Learning	'International Journal of Intelligent System and Application in Engineering (IJISAE))'	SVM, R.F.	Software metrics	accuracy rate of 0.87
4	Singh & Chhabra, [61]	"Improved software fault prediction using new code metrics and machine learning algorithms"	Machine Learning and New Code Metrics	'Journal of Computer Languages (Elsevier)'	SVM, Logistic Regression (L.R.), KNN, Decision Tree (D.T.), Multilayer Perceptron (MLP), Naïve Bayes (N.B.), Adaboost, Gradient Boost	13 Open Source Datasets (Ant – 1.6 and1.7, Xalan – 2.5 and 2.6, POI – 2.5and 3.0, Jedit- 4.1, Log4j-1.1, Camel 1.4 and 1.6, Xerces-1.3 and 1.4, Synapse – 1.2)	Ant 1.6 – L.R. result in 0.82411, Xalan- 2.5 Gradient Boost is 0.83035, Jedit -4.1 LR gives 0.86045, Log4j- 1.1 LR is 0.85497, camel 1.4, Xalan -2.6 LR result in 0.71770 and 0.83035 , Xerces and synapse SVM gives – 0.9782 and 0.77684
5	Ali et al., [27]	"BERT-based severity prediction of bug reports for the maintenance of mobile applications"	Deep Learning	'Journal of Systems and Software '	'Bidirectional Encoder Representations from Transformers' (BERT- SBR) based severity prediction of bug reports	*Hugging Face,* SentiWord Net	BERT – SBR result in Accuracy- 91.13, Precision- 91.02, Recall- 91.13, and F- Measure -91.03
6	Mamatha, R., Kumari, P. L. S., &	"Enhanced Software Defect Prediction Through	Machine Learning	'International Journal of Intelligent System and Application	Naive Bayes, Bagging, Boosting, and Grid Search	NASA	The result in the receiver operating characteris

	Sharada , A. [60]	Homogeneous Ensemble Models"		in Engineering (IJISAE)'			tics curve) ROC and Lift Curve show increased accuracy and recall rates.
7	Liu et al., [28]	"Revisiting Code Smell Severity Prioritization using learning to rank techniques"	Machine Learning based Comprehensive review	'Expert Systems with Applications (ELSEVIER)'	'Code Smell Severity Prioritization'(CSSP), 'Learning To Rank' (LTR) algorithm	God Class, Data Class, Feature Envy,	thorough comparison of the Bagging algorithm with 41 pointwise, four pairwise, and four listwise LTR techniques, yielding severity @20%
8	Wang et al. 2024[66]	"An empirical assessment of different word embedding and deep learning models for bug assignment"	Deep Learning	'Journal of Systems and Software on Science Direct'	'Word2Vec, GloVe, NextBug, ELMo, and BERT'	Eclipse JDT, GCC, Firefox	The top-k (k=1, 5, 10) accuracy and Mean Reciprocal Rank (MRR) were reported for each model.
9	Gomes et al. (2023[52]	"BERT- and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study"	Machine Learning	'Information and Software Technology'	KNN, NB, N.N., R.F., SVM	Freedesktop, GCC, Mozilla, Eclipse, Gnome, Wine HQ 2022	SVM is best in 3 datasets in which Mozilla gives better results than others, which is 61.5%
10	Panda & Nagwani, [53]	"An intuitionistic fuzzy representation-based software	'Fuzzy Similarity Measure-Based Software Bug	'Engineering Applications of Artificial Intelligence'	'IFSBSP model from different data sets'.	'Eclipse, 'Mozilla', 'Apache', and 'NetBeans'	accuracy ranging from 88.1% to 92.9% and F-measure

		bug severity prediction approach for imbalanced severity classes"	Severity Prediction' (FBSP)				ranging from 89.3% to 91.7%
11	Kukkar et al., [54]	"Bug severity classification in software using ant colony optimization-based feature weighting technique"	'Ant colony optimization' (ACO) based feature extraction technique	'Expert systems with applications'	'N.B., SVM, LR, DeepFM, and F-SVM methods'	'Eclipse', 'Mozilla', 'OpenFOAM', 'JBoss', and 'Firefox'	accuracy rates ranging from 85.73% to 89.38% for ACO-F-SVM, 78% to 80% for ACO-NB, 73% to 76% for ACO-SVM, and 92.67% to 97.27% for ACO-DeepFM
12	Olaleye et al., [55]	"Predictive Analytics and Software Defect Severity: A Systematic Review and Future Directions"	Machine Learning for SLR	'Hindawi Scientific Programming'	The preferred choice for NB	NASA, Apache, Mozilla, Promise, Eclipse, Azeem	52 primary studies that were selected based on a population, intervention, and outcome model
13	Tabassum et al., [34]	"Classification of Bugs in Cloud Computing Application	Machine Learning, Natural Language Processing	'Applied Sciences (MDPI)'	NB, DT, LR, RF	From Kaggle 2020	RF fared better 91.73 % accuracy rate

		s Using Machine Learning Techniques ”	(NLP), Cloud Computing				
14	Yoo et al., [64]	“Lite and Efficient Deep Learning Model for Bearing Fault Diagnosis Using the CWRU Dataset”	Deep Learning	‘Sensors (MDPI)’	Convolution Neural Network (CNN)	Case Western Reserve University (CWRU) Dataset	Accuracy of 64% achieved
15	Hazra et al., [56]	“Is This Bug Severe? A Text-Cum-Graph Based Model for Bug Severity Prediction”	Graph-Based Learning	‘Joint European Conference on Machine Learning and Knowledge Discovery 2022’	Graph attention network (GAT), Doc2Vec, Graph convolution network (Gcn), SBERT	Mozilla, Eclipse, GCC	Doc2Vec 7.585, Sbert 1.081, GCN 1.195, GAT 1.103, all result of 70% training set
16	Hamouri et al., [31]	“Predicting Bug Severity Using Machine Learning and Ensemble Learning Techniques ”	‘Machine Learning’	‘‘IEEE 14th International Conference on Information and Communication System “	K-Nearest Neighbors, Naïve Bayes, Neural Networks, Random Forest, Support Vector Machine	6 Open source datasets	accuracy that ranges from 93% to 95%

17	Meesad et al., [32]	"Machine Learning-Based Methods for Identifying Bug Severity Levels from Bug Reports"	Machine learning	'19 The International Conference on Computing and Information Technology '	L.R., R.F., SVM, LSTM	Core, Firefox, Thunderbird, and Bugzilla	L.R. – 0.8927, R.F. – 0.8871, SVM – 0.8915 conclude from all datasets L.R. gives better result
18	Hao et al., [63]	"A novel Vulnerability severity assessment method for source code based on a graph neural network"	Deep Learning	'Information and Software Technology (Elsevier ScienceDirect)'	CNN, CNN+SVM, GNN	National Vulnerability and Software Assurance Reference Dataset (SARD),	Accuracy CNN – 66.61, CNN + SVM - 70.81, GNN – 83.77 conclude that accuracy n increase by 10%
19	Shatnawi et al. [33]	"An Assessment of Eclipse Bugs' Priority and Severity Prediction Using Machine Learning"	Machine Learning	'International Journal of Communication Network and Information Security'	Ada Boosting, NB, R.F. , KNN, Linear SVM , Bagging,	Eclipse	The resulting range of F1 scores between 73 – 79 %, and Ada boosting and Bagging give accuracy much better than others, which is achieved near to100%

20	Kim & Yang, [57]	"Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm"	Deep Learning	'IEEE Access'	CNN	Eclipse and Mozilla	90.62% of the F-measure and 93.22% of Mozilla
21	Bibyan et al. [58]	"Latent Dirichlet Allocation (LDA) Based on Automated Bug Severity Prediction Model"	LDA	'Proceedings of Data Analytics and Management'	N.B., Multinomial naive base (MNB),k-nearest neighbour (k-NN), RF, DT, artificial neural network (ANN) and SVM	Eclipse	Based on component IDE, gives better results with MNB classifier from all classes 0.921accuracy with F1 score of 0.875 and recall is 0.921
22	Ahmed et al.,[67]	"CaPBug-A Framework for Automatic Bug	CaPBug	'IEEE Access'	NB, RF, DT, LR	Mozilla and Eclipse	CaPBug
		Categorization and Prioritization Using NLP and Machine Learning Algorithms"	Framework				Framework R.F. achieved 94%, L.R. – 84.33%, N.B. - 83.29%n accuracy of 90.43%, D.T. – 88.94%
23	Johnson et al., [62]	"Optimized ensemble machine learning model for software bug prediction"	Machine Learning	'Innovations in System and Software Engineering'	D.T., L.R., SVM, NB, N.N., Ensemble (LR+ et al.)	Eight different NASA datasets	the overall accuracy of the ensemble model is 97.8%; the ensemble model achieved a significant increase in prediction accuracy from 96.7-97.8% for the un-

							vectorized dataset to the vectorized dataset.
24	Sepahvand et al.,[68]	"Using word embedding and convolution neural networks for bug triaging by considering design flaws"	Deep Learning	'Science of Computer Programming (Elsevier ScienceDirect)'	CNN	ITS and GitHub	efficiency of the algorithm varies from 55% to 78%, and accuracy of 75% or more in different datasets
25	Dharmakeerthi et al., [69]	"Bug priority prediction using deep ensemble approach"	Deep Learning	"Applied Soft Computing"	Bugzilla	Ensemble approach give efficiency of 79. 18 %	
26	Kumari et al., [70]	"Prioritization of Software Bugs Using Entropy-Based Measures „	Deep Learning	"Software Evolution and Process"	Eclipse, Mozilla, OpenOffice	Employed approaches f measures maximum – 46, 35, 28, 11, 15, 4, 3 and 1.	
27	Sarawan et al.,[71]	"Analyzing Hybrid Feature Representations for Improved Multiclass Bug Severity Classification „	Machine Learning	"Engineering Technology and Applied Science Research"	Online	Accuracy range between 65.95 % - 66.15%.	
28	Khleel et al., [72]	"Ensemble-Based Machine Learning Algorithms Combined with Near Miss Method for Software Bug Prediction"	Ensemble of machine and deep learning	"International Journal of Networked and Distributed Computing"	Five public datasets	Range is 81% - 97% for all datasets ensemble models	
29	Cao and Cui [73]	**"**Complex Network Neural Dynamics Framework for Automated Software Bug Triaging „	Deep Learning	"IEEE Access"	Eclipse, Mozilla, and GCC	accuracies of 80.52%, 79.67%, and 79.16% on the three datasets	
30	Zhang et al.,[74]	**"**BTAL: An imbalance software bug report triage approach based on BERT-TextCNN „	Deep Learning	"Information and Software Technology"	GCC, NetBeans, Eclipse, Mozilla, and OpenOffice,	BTAL approach used by triagers	

Bug severity prediction in software development is improving significantly, which improves the efficacy and efficiency of bug management and decision-making procedures.

7. Conclusion

Software bug reports are an essential tool for discovering and communicating issues of varied severity. It is imperative to manage software defects effectively. However, manual assignment of severity levels can introduce subjective biases and inaccuracies. To mitigate these issues, there has been a growing interest in exploring methods for automatically predicting severity levels.

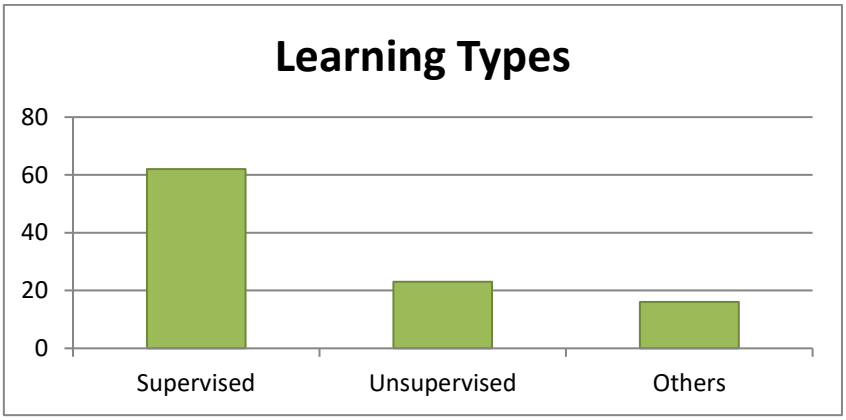


Figure 6: Research Direction distribution based on Learning Type

A variety of methods, including ensemble and hybrid learning, deep learning, and machine learning, have been used to forecast the severity of bugs in recent times. Nearly two-thirds of the studies have employed supervised learning, as shown in Figure 6. Deep learning has significant potential for enhancing severity prediction accuracy by leveraging bug reports' complex, multi-dimensional nature.

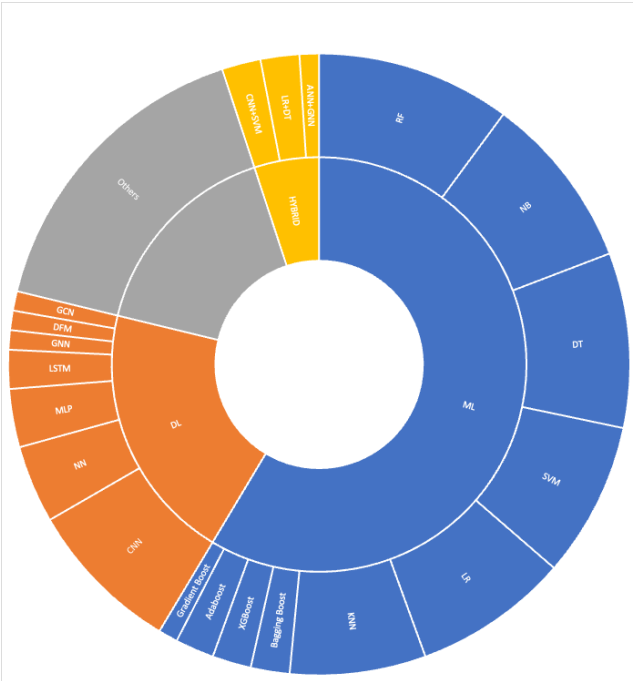


Figure 7: Current Research Directions

In the prediction of bug severity, deep learning accounts for 23%. In comparison, machine learning comprises 61%, with the remaining 10% attributed to other methodologies and 6 % to a hybrid of some techniques, as shown in Figure 7. In machine learning, supervised learning techniques have been used. Figure 8 shows the accuracy rate along with the publications is shown below .

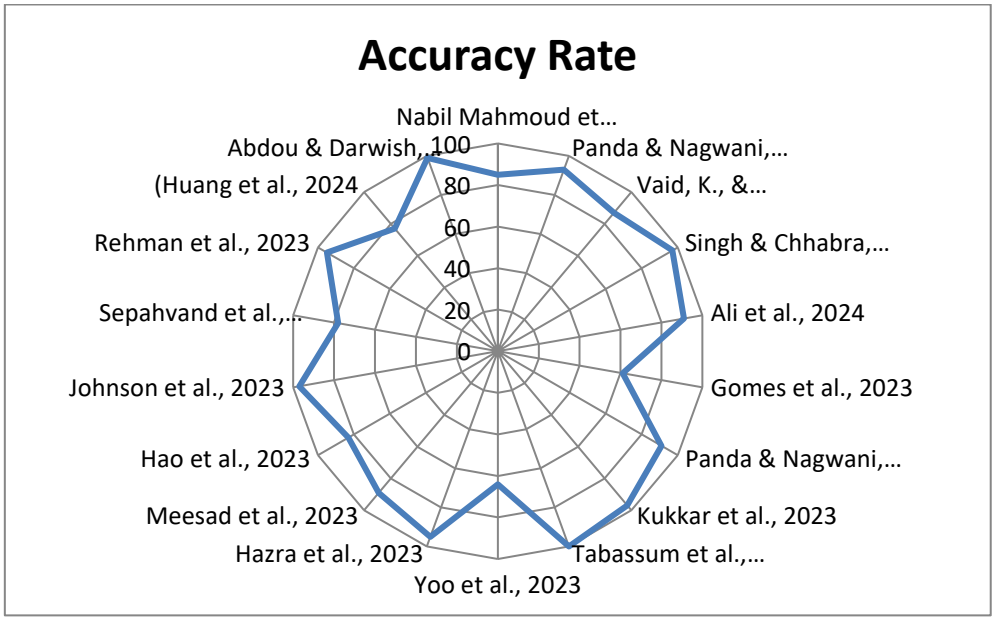


Figure 8: Reported Accuracy rate along with publication in 2023-2024 for bug severity.

Various techniques have been proposed, from traditional machine learning algorithms to more advanced deep learning methods. While traditional approaches have shown promise, exploring deep learning techniques still needs to be improved in this context. However, its comprehensive application in this field has yet to be fully realized, presenting an opportunity for further research and advancement. The pursuit of automated severity prediction methods significantly improves the efficiency and effectiveness of bug management processes, ultimately leading to more robust and reliable software systems. Continued exploration and innovation in this area will be essential for meeting the evolving challenges of software maintenance and development.

8. References

[1] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles, "Towards a simplification of the bug report form in Eclipse," in *Proc. Int. Working Conf. Mining Softw. Repositories*, 2008, pp. 145–148.

[2] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *Proc. IEEE Int. Conf. Software Maintenance*, 2008, pp. 346–355.

[3] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Proc. 7th IEEE Working Conf. Mining Softw. Repositories*, 2010, pp. 1–10.

[4] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Proc. 15th Eur. Conf. Software Maintenance and Reengineering*, 2011, pp. 249–258.

[5] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. D. C. Machado, T. F. Vale, E. S. de Almeida, and S. R. D. L. Meira, "Challenges and opportunities for software change request repositories: A systematic mapping study," *Journal of Software: Evolution and Process*, vol. 26, no. 7, pp. 620–653, 2014.

[6] L. A. F. Gomes, R. D. S. Torres, and M. L. Côrtes, "Bug report severity level prediction in open source software: A survey and research opportunities," *Information and Software Technology*, vol. 115, pp. 58–78, 2019.

[7] C.-Z. Yang, C.-C. Hou, W.-C. Kao, and I.-X. Chen, "An empirical study on improving severity prediction of defect reports using feature selection," in *Proc. 19th Asia-Pacific Software*

Engineering Conf., Hong Kong, China, Dec. 4–7, 2012, vol. 1, pp. 240–249.

[8] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proc. 34th Int. Conf. Software Engineering (ICSE)*, Zurich, Switzerland, Jun. 2–9, 2012, pp. 25–35.

[9] G. Yang, T. Zhang, and B. Lee, "An emotion similarity-based severity prediction of software bugs: A case study of open source projects," *IEICE Transactions on Information and Systems*, pp. 2015–2026, 2018.

[10] M. Sharma, M. Kumari, R. K. Singh, and V. B. Singh, "Multiattribute based machine learning models for severity prediction in cross-project context," in *Transactions on Petri Nets and Other Models of Concurrency XV*. Berlin/Heidelberg, Germany: Springer, 2014, pp. 227–241.

[11] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang, "ELBlocker: Predicting blocking bugs with ensemble imbalance learning," *Information and Software Technology*, vol. 61, pp. 93–106, 2015.

[12] A. F. Otoom, D. Al-Shdaifat, M. Hammad, and E. E. Abdallah, "Severity prediction of software bugs," in *Proc. 7th Int. Conf. Information and Communication Systems (ICICS)*, Irbid, Jordan, Apr. 5–7, 2016, pp. 92–95.

[13] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces," in *Proc. 26th Int. Conf. Computer Science and Software Engineering (CASCON '16)*, Toronto, ON, Canada, Oct. 31–Nov. 2, 2016, pp. 96–105.

[14] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.

[15] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *Journal of Software: Evolution and Process*, vol. 28, pp. 150–176, 2016.

[16] C.Z. Yang, K.-Y. Chen, and A.-H. Dao, "Bug severity assessment based on weighted multi-facet features with particle swarm optimization," *Software Quality Journal*, year not provided (update year if needed).

[17] C.Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, "Improving severity prediction on software bug reports using quality indicators," in *Proc. 5th IEEE Int. Conf. Software Engineering and Service Science (ICSESS)*, Beijing, China, 2014, pp. xx–xx. *(Add pages if available.)*

[18] A. Kukkar, R. Mohana, A. Nayyar, J. Kim, B.-G. Kang, and N. Chilamkurti, "A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting," *Sensors*, vol. 19, no. 2964, 2019.

[19] E. E. Abdallah, A. Aljammal, A. F. Otoom, M. Hammad, and D. Al Shdaifat, "Automated labelling and severity prediction of software bug reports," *International Journal of Computer Science and Engineering*, vol. 19, p. 330, 2019.

[20] W. Y. Ramay, Q. Umer, X. C. Yin, C. Zhu, and I. Illahi, "Deep neural network-based severity prediction of bug reports," *IEEE Access*, vol. 7, pp. 46846–46857, 2019.

[21] M. Sharma, M. Kumari, and V. B. Singh, "Multi-attribute dependent bug severity and fix time prediction modelling," *International Journal of System Assurance Engineering and Management*, vol. 10, pp. 1328–1352, 2019.

[22] K. K. Sabor, M. Hamdaqa, and A. Hamou-Lhadj, "Automatic prediction of the severity of bugs using stack traces and categorical features," *Information and Software Technology*, vol. 123, 106205, 2020.

[23] G. Yang, S. Baek, J.-W. Lee, and B. Lee, "Analyzing emotion words to predict severity of software bugs: A case study of open source projects," in *Proc. Symposium on Applied Computing (SAC '17)*, Marrakech, Morocco, Apr. 4–6, 2017, pp. 1280–1287.

[24] S. Ahlawat and D. Khurana, "CNN based software bug severity prediction," in *Progressive Computational Intelligence, Information Technology and Networking*, CRC Press, 2025, pp. 646–649.

[25] K. Vaid and U. Ghose, "Machine learning based predictive analysis of software bug severity and priority," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 15s, pp. 249–256, 2024.

[26] R. R. Panda and N. K. Nagwani, "Software bug severity and priority prediction using SMOTE

and intuitionistic fuzzy similarity measure," *Applied Soft Computing*, vol. 150, p. 111048, 2024.

[27] A. Ali, Y. Xia, Q. Umer, and M. Osman, "BERT-based severity prediction of bug reports for the maintenance of mobile applications," *Journal of Systems and Software*, vol. 208, p. 111898, 2024.

[28] L. Liu *et al*., "Revisiting code smell severity prioritization using learning to rank techniques," *Expert Systems with Applications*, vol. 249, p. 123483, 2024.

[29] R. R. Panda and N. K. Nagwani, "Software bug categorization technique based on fuzzy similarity," in *Proc. IEEE 9th Int. Conf. Advanced Computing (IACC)*, 2019, pp. 1–6.

[30] A. Hamdy and A. El-Laithy, "Semantic categorization of software bug repositories for severity assignment automation," in *Integrating Research and Practice in Software Engineering*, S. Jarzabek, A. Poniszewska-Marańda, and L. Madeyski, Eds., vol. 851, *Studies in Computational Intelligence*. Cham: Springer, 2020, pp 22-33.

[31] S. K. Hamouri *et al*., "Predicting bug severity using machine learning and ensemble learning techniques," in *Proc. 14th Int. Conf. Information and Communication Systems (ICICS)*, 2023, pp. 1–6.

[32] P. Meesad, S. Sodsee, W. Jitsakul, and S. Tangwannawit, Eds., *Proc. 19th Int. Conf. Computing and Information Technology (IC2IT 2023)*, vol. 679. Springer, 2023.

[33] M. Q. Shatnawi and B. Alazzam, "An assessment of Eclipse bugs' priority and severity prediction using machine learning," *International Journal of Communication Networks and Information Security*, vol. 14, no. 1, pp. 62–69, 2022.

[34] N. Tabassum *et al*., "Classification of bugs in cloud computing applications using machine learning techniques," *Applied Sciences*, vol. 13, no. 5, p. 2880, 2023.

[35] R. S. Rao, S. Dewangan, A. Mishra, and M. Gupta, "A study of dealing class imbalance problem with machine learning methods for code smell severity detection using PCA-based feature selection technique," *Scientific Reports*, vol. 13, no. 1, p. 16245, 2023.

[36] S. Dewangan, R. S. Rao, A. Mishra, and M. Gupta, "A novel approach for code smell detection: An empirical study," *IEEE Access*, vol. 9, pp. 162869–162883, 2021.

[37] S. K. Pandey and A. K. Tripathi, "An empirical study towards dealing with noise and class imbalance issues in software defect prediction," *In Review* (preprint), 2021. doi: 10.21203/rs.3.rs-549406/v1.

[38] Y. Fan, X. Xia, D. Lo, and A. E. Hassan, "Chaff from the wheat: Characterizing and determining valid bug reports," *IEEE Transactions on Software Engineering*, vol. 46, no. 5, pp. 495–525, May 2020.

[39] F. Pecorelli, F. Palomba, F. Khomh, and A. De Lucia, "Developer-driven code smell prioritization," in *Proc. 17th Int. Conf. Mining Software Repositories*, 2020, pp. 220–231.

[40] Z. Huang *et al*., "Aligning XAI explanations with software developers' expectations: A case study with code smell prioritization," *Expert Systems with Applications*, vol. 238, p. 121640, 2024.

[41] S. Dewangan, R. S. Rao, and P. S. Yadav, "Dimensionally reduction based machine learning approaches for code smells detection," in *Proc. Int. Conf. Intelligent Controller and Computing for Smart Power (ICICCSP)*, 2022, pp. 1–4.

[42] L. Rehman , "Long-lived bugs prediction using machine learning approaches," OSF Preprint, doi: 10.17605/OSF.IO/T2Z75.

[43] S. R. O. Peralta *et al*., "Analysis of bug report qualities with fixing time using a Bayesian network," in *Proc. 27th Int. Conf. Evaluation and Assessment in Software Engineering*, 2023, pp. 235–240.

[44] L. Chmielowski, P. Konstantynov, R. Luczak, M. Kucharzak, and R. Burduk, "A novel method for software bug report assignment," *Reliability: Theory & Applications*, vol. 18, no. 2, pp. 579–588, 2023.

[45] N. K. Nagwani and J. S. Suri, "An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review," *International Journal of Information Management Data Insights*, vol. 3, no. 1, p. 100153, 2023.

[46] R. Agrawal and R. Goyal, "Developing bug severity prediction models using word2vec," *International Journal of Cognitive Computing in Engineering*, vol. 2, pp. 104–115, 2021.

[47] S. F. A. Zaidi and C.-G. Lee, "Learning graph representation of bug reports to triage bugs using graph convolution network," in *Proc. Int. Conf. Information Networking (ICOIN)*, 2021, pp. 504–507.

[48] S. Gujral, G. Sharma, S. Sharma, and Diksha, "Classifying bug severity using a dictionary-based approach," in *Proc. Int. Conf. Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, 2015, pp. 599–602.

[49] I. Kaur and A. Kaur, "Comparative analysis of software fault prediction using various categories of classifiers," *International Journal of System Assurance Engineering and Management*, vol. 12, no. 3, pp. 520–535, 2021.

[50] S. Mehta and K. S. Patnaik, "Improved prediction of software defects using ensemble machine learning techniques," *Neural Computing and Applications*, vol. 33, no. 16, pp. 10551–10562, 2021.

[51] T. Sharma, V. Efstathiou, P. Louridas, and D. Spinellis, "Code smell detection by deep direct-learning and transfer-learning," *Journal of Systems and Software*, vol. 176, p. 110936, 2021.

[52] L. Gomes, R. Da Silva Torres, and M. L. Côrtes, "BERT- and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: A comparative study," *Information and Software Technology*, vol. 160, p. 107217, 2023.

[53] R. R. Panda and N. K. Nagwani, "An intuitionistic fuzzy representation-based software bug severity prediction approach for imbalanced severity classes," *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106110, 2023.

[54] A. Kukkar, Y. Kumar, A. Sharma, and J. K. S. Sandhu, "Bug severity classification in software using ant colony optimization-based feature weighting technique," *Expert Systems with Applications*, vol. 230, p. 120573, 2023.

[55] T. O. Olaleye, O. T. Arogundade, S. Misra, A. Abayomi-Alli, and U. Kose, "Predictive Analytics and Software Defect Severity: A Systematic Review and Future Directions," *Scientific Programming*, vol. 2023, pp. 1–18, 2023.

[56] R. Hazra, A. Dwivedi, and A. Mukherjee, "Is This Bug Severe? A Text-Cum-Graph Based Model for Bug Severity Prediction," in *Machine Learning and Knowledge Discovery in Databases*, vol. 13718, pp. 236–252, Springer, 2023.

[57] J. Kim and G. Yang, "Bug Severity Prediction Algorithm Using Topic-Based Feature Selection and CNN-LSTM Algorithm," *IEEE Access*, vol. 10, pp. 94643–94651, 2022.

[58] R. Bibyan, S. Anand, and A. Jaiswal, "Latent Dirichlet Allocation (LDA) Based Automated Bug Severity Prediction Model," in *Proceedings of Data Analytics and Management*, vol. 90, pp. 363–377, Springer, 2022.

[59] A. N. Mahmoud, A. Abdelaziz, V. Santos, and M. M. Freire, "A proposed model for detecting defects in software projects," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 1, p. 290, 2024.

[60] R. Mamatha, P. L. S. Kumari, and A. Sharada, "Enhanced Software Defect Prediction Through Homogeneous Ensemble Models," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 4s, pp. 676–684, 2023.

[61] M. Singh and J. K. Chhabra, "Improved software fault prediction using new code metrics and machine learning algorithms," *Journal of Computer Languages*, vol. 78, p. 101253, 2024.

[62] F. Johnson, O. Oluwatobi, O. Folorunso, A. V. Ojumu, and A. Quadri, "Optimized ensemble machine learning model for software bugs prediction," *Innovations in Systems and Software Engineering*, vol. 19, no. 1, pp. 91–101, 2023.

[63] J. Hao, S. Luo, and L. Pan, "A novel vulnerability severity assessment method for source code based on a graph neural network," *Information and Software Technology*, vol. 161, p. 107247, 2023.

[64] Y. Yoo, H. Jo, and S.-W. Ban, "Lite and Efficient Deep Learning Model for Bearing Fault Diagnosis Using the CWRU Dataset," *Sensors*, vol. 23, no. 6, p. 3157, 2023.

[65] A. Abdou and N. Darwish, "Severity classification of software code smells using machine learning techniques: A comparative study," *Journal of Software: Evolution and Process*, vol. 36, no. 1, p. e2454, 2024.

[66] R. Wang, X. Ji, S. Xu, Y. Tian, S. Jiang, and R. Huang, "An empirical assessment of different word embedding and deep learning models for bug assignment," *Journal of Systems and Software*, vol. 210, p. 111961, 2024.

[67] H. A. Ahmed, N. Z. Bawany, and J. A. Shamsi, "CaPBug-A Framework for Automatic Bug Categorization and Prioritization Using NLP and Machine Learning Algorithms," *IEEE Access*, vol. 9, pp. 50496–50512, 2021.

[68] R. Sepahvand, R. Akbari, B. Jamasb, S. Hashemi, and O. Boushehrian, "Using word embedding and convolution neural network for bug triaging by considering design flaws," *Science of Computer Programming*, vol. 228, p. 102945, 2023.

[69] P. G. S. M. Dharmakeerthi, R. A. H. M. Rupasingha, and B. T. G. S. Kumara, "Bug priority prediction using deep ensemble approach," *Applied Soft Computing*, vol. 175, p. 113098, 2025.

[70] M. Kumari, R. Singh, and V. B. Singh, "Prioritization of software bugs using entropy-based measures," *Journal of Software: Evolution and Process*, vol. 37, no. 2, p. e2742, 2025.

[71] K. Sarawan *et al*., "Analyzing hybrid feature representations for improved multiclass bug severity classification," *Engineering, Technology & Applied Science Research*, vol. 15, no. 4, pp. 24561–24569, 2025.

[72] K. Sarawan *et al*., "Analyzing hybrid feature representations for improved multiclass bug severity classification," *Engineering, Technology & Applied Science Research*, vol. 15, no. 4, pp. 24561–24569, 2025.

[73] H. Cao and M. Cui, "Complex network neural dynamics framework for automated software bug triaging," *IEEE Access*, 2025.

[74] Y. Zhang *et al*., "BTAL: An imbalance software bug report triage approach based on BERT-TextCNN," *Information and Software Technology*, vol. 183, p. 107731, 2025.