International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Machine Learning and Just-in-Time Strategies for Effective Bug Tracking in Software Development

[1]Veena Jadhav, [2]Dr. Prakash Devale, [3]Dr.Rohini B. Jadhav, [4]Ranjeet Vasant Bidwe, [5]Madhavi Mane, [6]Prajakta Pawar

Submitted: 24/09/2023 **Revised**: 16/11/2023 **Accepted**: 28/11/2023

Abstract: Effective bug tracking and resolution are crucial for maintaining software quality and ensuring timely project delivery in the constantly changing field of software development. This research paper introduces an innovative method that combines machine learning techniques with just-in-time (JIT) strategies to improve bug tracking and resolution processes. In the study JM1 dataset is used for software defect prediction. This work also introduces a comprehensive feature engineering methodology to extract relevant information from the dataset. The study proposed a hybrid model that incorporate Random Forest and Support Vector Machine (SVM) classifiers, to forecast and rank software defects according to different bug attributes. The proposed model exhibits a remarkable accuracy of 98.79%, thereby demonstrating its efficacy in precisely detecting and prioritizing bugs. The exceptional level of precision is credited to the robust feature engineering method, which considers complexity metrics and historical defect density. The research highlights the importance of promptly addressing newly reported bugs by implementing just-in-time (JIT) principles in bug tracking practices. This involves assigning and prioritizing bugs in real-time within the current development cycle. The integration of JIT and machine learning optimizes the software development process by reducing delays, speeding up problem-solving, and improving overall efficiency. The research findings offer valuable insights for software development teams aiming to enhance the efficiency of their bug tracking procedures. The combination of the Random Forest and SVM model, enhanced by JIT strategies, offers a highly effective framework for guaranteeing software quality and timely project completion in the rapidly evolving field of software development. This research provides a current and pragmatic method for staying ahead of software defects, as the software industry continues to progress.

Keywords: *Bug tracking, Just in Time (JIT), Machine Learning, Software fault prediction, Software development.*

1. Introduction

Software development is a complex and ever-changing process, marked by ongoing iterations, evolving requirements, and a persistent pursuit of excellence. Software development projects must utilize robust bug-tracking methodologies to efficiently identify, manage, and rectify software defects. This introduction provides an in-depth exploration of bug tracking, encompassing its historical origins, current difficulties, and the inventive approaches that are transforming software development methodologies[1], [2].

In the past, during the initial stages of software development, the process of tracking and identifying software defects was primarily done manually and required a significant amount of effort. Developers utilized basic tools, such as spreadsheets, emails, or handwritten bug reports, to record and oversee software defects. The conventional method had certain limitations, particularly in regards to clarity, communication, and effectiveness. The dependence on manual procedures frequently led to a lack of transparency, impeding timely problem resolution and causing significant accumulations of unresolved defects. Conventional bug-tracking methodologies were hindered by their dependence on human intervention, which prevented them from adapting to the ever-changing requirements of modern software development[3]–[5].

Conventional bug-tracking methods posed numerous inherent difficulties. Software development teams faced various limitations, including delayed bug identification, insufficient prioritization, and challenges in tracking historical defect patterns. These constraints frequently

[1]*Assistant Professor, Department of Computer Engineering, Bharati Vidyapeeth(Deemed to be University) College of Engineering, Pune, Maharashtra, India*
vjjadhav@bvucoep.edu.in
Professor,
[2]*Department of Information Technology, Bharati Vidyapeeth(Deemed to be university) College of Engineering, Pune, Maharashtra, India*
prdevale@bvucoep.edu.in
[3]*Associate Professor, Department of Information Technology, Bharati Vidyapeeth(Deemed to be university) College of Engineering, Pune, Maharashtra, India*
rbjadhav@bvucoep.edu.in
[4]*Symbiosis Institute of Technology, Pune (SIT), Symbiosis International (Deemed)University (SIU), Lavale, Pune, Maharashtra, India*
ranjeet.bidwe@sitpune.ed.in
[5]*Assistant Professor, Department of Computer Engineering, Bharati Vidyapeeth(Deemed to be University) College of Engineering, Pune, Maharashtra, India*
mmmane@bvucoep.edu.in
[6]*Assistant Professor, Bharati Vidyapeeth's College of Engineering Lavale Pune, Maharashtra, India*
pawar.prajakta@bharatividyapeeth.edu

led to subpar software quality, prolonged project schedules, and dissatisfied stakeholders.

Software development teams adopted Just-in-Time (JIT) methodologies as a solution to the limitations of traditional bug tracking. JIT principles, such as Agile and Scrum, promote the use of iterative development, continuous integration, and adaptability. These methodologies aim to divide the development process into smaller cycles or sprints, each of which involves a concentrated and time-limited development effort. JIT methodologies prioritize adaptability and promptness, allowing development teams to promptly rectify defects as they arise. JIT defect prediction and bug tracking enhance these principles by seamlessly incorporating the process of identifying, prioritizing, and resolving defects into the development workflow, thereby aligning it with project objectives and priorities[6], [7].

Agile development methodologies adopt a customer-focused and cooperative approach. Their primary focus is on delivering functional software in iterative cycles, and agile teams are highly adaptable to changing requirements. These iterative practices inherently result in the early identification and resolution of defects, as there is a significant emphasis on consistently maintaining software quality.

Scrum, a widely used Agile methodology, mandates frequent sprint cycles, with each one resulting in a potentially deliverable product increment. Scrum incorporates bug tracking by assigning distinct roles and conducting specific ceremonies to handle the identification, ranking, and resolution of defects. This strategic approach guarantees that defects are not overlooked but are dealt with as an essential component of the development process.

The concept of JIT defect prediction expands upon JIT principles by actively predicting and preventing defects in advance. It accomplishes this by detecting possible problems at an early stage in the development process using historical data and predictive analytics. Through comprehending historical defect patterns, development teams can proactively implement measures to mitigate defects prior to them reaching a critical state[8]. JIT bug tracking applies the principles of JIT defect prediction. It guarantees that recently reported defects are promptly evaluated and allocated within the ongoing development phase. The approach reduces the occurrence of development obstacles, expedites the resolution of problems, and simplifies the software development process.

The domain of machine learning has introduced a revolutionary period in the process of bug tracking. Machine learning methods provide data-based insights, predictive abilities, and automated procedures that have greatly enhanced the bug-tracking process in software development. Machine learning has been applied to various aspects of bug tracking[9]. These encompass bug prediction, issue classification, automatic assignment, and the estimation of defect resolution durations. Machine learning models enhance the accuracy and efficiency of bug-tracking processes by utilizing pertinent features derived from bug reports, historical data, and code complexity metrics[10].

Many researchers proposed machine learning models and algorithms to forecast, categorize, and rank bugs. These models utilize sophisticated feature engineering techniques to extract significant information from the data. They employ machine learning algorithms to make precise predictions and enhance the bug resolution process. The abundance of available research has resulted in the creation of models that are becoming more proficient at detecting and ranking software defects.

The JM1 dataset plays a crucial role in advancing research on software defect prediction. This dataset is well-known in the field of software defect prediction and provides a comprehensive and valuable source of information for evaluating and verifying machine learning models. The collection consists of a wide range of software metrics and characteristics associated with bugs. This allows researchers to investigate the impact of feature engineering and modeling techniques on the accuracy of bug tracking.

Feature engineering plays a crucial role in improving the predictive abilities of machine learning models. The process entails extracting, transforming, and selecting features from the dataset to enhance the model's capacity to detect and prioritize software defects. Feature engineering involves the development of complexity metrics, historical defect density, and other domain-specific attributes, which help in accurately predicting bugs.

This study presents a novel hybrid model that combines the advantages of Random Forest and Support Vector Machine (SVM) classifiers. The hybrid model combines the ensemble learning capabilities of Random Forest with the nonlinear classification abilities of SVM. The purpose of this amalgamation is to provide highly precise bug predictions. The initial findings indicate a remarkable accuracy rate of 98.79%, highlighting the model's capacity to offer accurate bug predictions.

The incorporation of JIT strategies into bug tracking is crucial for this research. This involves promptly evaluating and assigning newly reported bugs in real-time, within the constraints of the ongoing development cycle. By employing a strategic real-time approach, the

occurrence of development bottlenecks is reduced, the resolution of issues is expedited, and the overall software development process becomes more efficient.

The research presented in this paper is characterized by its multifaceted nature. The main goals of this are two-fold. The primary objective is to investigate the utilization of machine learning models in bug tracking, specifically emphasizing the hybrid Random Forest + SVM model. Furthermore, it aims to assess the influence of JIT strategies on the tracking and resolution of bugs in the software development process. The research objectives focus on acquiring a more profound comprehension of the complex interaction among machine learning, JIT principles, and conventional bug-tracking methodologies.

This study has important implications for software development teams and researchers who are aiming to improve their bug-tracking and resolution processes. This research explores the combination of machine learning and JIT strategies to uncover valuable insights that can enhance bug tracking, minimize development bottlenecks, and ultimately enhance software quality. Understanding and applying these strategies is crucial in the constantly changing field of software development, as it ensures the delivery of top-notch software within specified time and financial limitations.

Literature Review

The bug tracking landscape in software development is continuously evolving, as development teams strive for more efficient and precise methods to manage defects. This literature review examines a range of important research papers that shed light on various aspects of bug tracking, with a specific focus on the incorporation of artificial intelligence (AI) and machine learning (ML) methods. These papers explore the difficulties and possibilities in bug prediction, analysis of real-world bug reports, AI frameworks for bug triaging, and the consequences of mislabeled data on just-in-time defect prediction models. The ongoing evolution of software development practices has led to valuable insights from studies that can enhance the accuracy and efficiency of bug-tracking processes.

R. Ferenc et al.[11] explores the utilization of deep learning methods to forecast software defects using static code metrics. The authors intend to utilize deep learning to enhance the accuracy and efficiency of bug detection during the software development process. This research introduces a captivating aspect to bug prediction, potentially transforming our approach to defect identification. M. Laiq et al.[12] employ an empirical methodology to understand the domain of erroneous bug reports. It accomplishes this by performing a comprehensive analysis, with a specific emphasis on practical industrial situations. The paper examines the attributes and sources of inaccurate bug reports, providing valuable insights into the difficulties encountered by software development teams and suggesting ways to enhance bug reporting and tracking procedures. N. K. Nagwani et al.[13] present a thorough examination of an artificial intelligence framework designed for the process of software bug triaging. This text not only examines the present level of advancement but also foresees forthcoming obstacles. The authors present a roadmap for dealing with the complexities of modern software development by analyzing the development of bug triaging and the incorporation of AI.

S. Albahli's[14] research centers on the complex task of forecasting software defects, taking into account the level of effort needed to address them. The study tackles the practical problem of accurately predicting bugs and considering the necessary resources for bug fixing by implementing a sophisticated deep ensemble learning approach. This work is extremely valuable in improving the effectiveness of bug prediction models. A. Kukkar et al.[15] present a programmer recommendation model that assists in accurate software bug management. The model utilizes “Ant Colony Optimization” (ACO) to offer programmers suggestions on efficiently handling and resolving software defects. This innovation serves as a connection between recommendation systems and bug management, ultimately enhancing the quality of software development. H. Xu et al.[16] explore the pivotal matter of data sampling for just-in-time defect prediction models. The researchers investigate different sampling strategies to address the issue of imbalanced data in bug prediction. The paper enhances the dependability and precision of defect prediction models by identifying efficient sampling techniques.

L. Jonsson et al.[17] investigate the automated allocation of bugs in large-scale industrial software development, with a specific focus on the practical difficulties encountered. The study tackles the intricate task of assigning reported bugs to the appropriate developers by utilizing ensemble-based ML techniques. This work aims to improve bug management in industrial contexts by optimizing the allocation of development resources. The study conducted by G. Rodriguez-Perez et al.[18] focuses on the influence of external bugs on just-in-time bug prediction models. The research examines the impact of external factors on the accuracy of bug prediction models, offering valuable insights into the practical difficulties of bug tracking, particularly in extensive open-source projects such as OpenStack. Y. Fan et al.[19] thoroughly examine the consequences of mislabeled changes caused by the SZZ algorithm, which is widely used, on just-in-time defect prediction. The paper emphasizes the importance of precise labeling in bug prediction models

and investigates the impact of mislabeled data on the dependability of predictions. This study highlights the importance of ensuring strong data accuracy in bug prediction research.

İ. Yazic et al.[20] conducted an extensive survey on the "Utilization of Artificial Intelligence and Machine Learning in Prospective Mobile Network-Enabled Systems". This study examines the crucial significance of artificial intelligence (AI) and machine learning (ML) in influencing the future development of mobile networks. Their research delves into various applications, emphasizing the continuously growing influence of these technologies on the advancement of mobile networks. D. Wang et al.[21] explores the complex processes of bug reproduction and localization, providing insight into the techniques and methods used in these crucial stages of bug management. This paper offers a valuable resource for comprehending the complexities of bug management practices. R. Ferenc et al.[22] introduce a comprehensive dataset called "Public Unified Bug Dataset for Java" and evaluate its performance in terms of metrics and bug prediction. This dataset functions as a standard for assessing bug prediction models and their corresponding metrics. It provides a standardized basis for researchers to evaluate and improve the precision of their bug prediction methods.

The study conducted by Z. Chen et al.[23] examines the alterations in dynamic feature code that occur during bug fixes, with a specific emphasis on evaluating the advantages and drawbacks of Python's dynamic features. Their research investigates the impact of bug fixes on the dynamic language features of Python, offering insights into the potential trade-offs and benefits of dynamic language elements in bug management. Q. Huang et al.[24] make a contribution to the field with their work titled "Revisiting Supervised and Unsupervised Models for Effort-Aware JIT Defect Prediction". The study rigorously evaluates the efficacy of supervised and unsupervised models in accurately predicting defects, with a specific emphasis on comprehending the amount of effort needed for timely defect prediction.

The analyzed research papers emphasize the complex nature of bug tracking in software development and emphasize the significance of integrating AI and ML methodologies into these processes. These papers provide valuable insights into understanding and enhancing bug tracking practices, ranging from deep learning methods for bug prediction to data-driven analyses of bug reports in industrial settings. The frameworks outlined in these studies, such as AI-powered bug triaging and recommendation models, have the capacity to revolutionize the management of defects in software development projects. Furthermore, the significance of data quality and its influence on defect prediction models cannot be overemphasized. These research papers are instrumental in advancing bug tracking strategies, leading to the development of superior software products.

2. Methodology

i. Dataset

The JM1 dataset, a leading software defect prediction resource, helps us understand software quality and defect management[25]. The dataset contains extensive historical Java project data on software metrics and defects. This provides researchers and practitioners with a standardized and comprehensive dataset to develop and test machine learning models for software defect prediction. The JM1 dataset includes code size, complexity, and maintainability metrics that reveal software quality. LOC (Lines of Code), NOM (Number of Methods), and CYCLO (Cyclomatic Complexity) are essential for understanding the codebase's structure. Defect prediction models must also be trained and evaluated using defect-related characteristics like module defects.

ii. Data Processing

a. Data normalization:

Data normalization guarantees that numerical features are uniformly scaled, preventing certain features from overpowering others in machine learning algorithms. The metrics "LOC" and "NOM" have significantly different numerical ranges. The variable "LOC" can have a wide range of values, spanning from hundreds to thousands, whereas the variable "NOM" typically ranges from a few to several hundred. By implementing Min-Max scaling, both features are transformed to a standardized scale ranging from 0 to 1, guaranteeing that no feature exerts disproportionate influence on the model. This enables a just and precise comparison and modeling of the dataset.

iii. Feature Engineering

a. Feature Selection

Feature selection is the process of identifying the most pertinent attributes from a dataset, while disregarding less informative or redundant features. The techniques employed in this study include recursive feature elimination (RFE) with a Random Forest algorithm to assess the significance of each feature as shown in fig.1. Metrics such as "LOC (Lines of Code)," "NOM (Number of Methods)," and "CYCLO (Cyclomatic Complexity)" have a significant influence on defect prediction, whereas other metrics have a lesser impact.

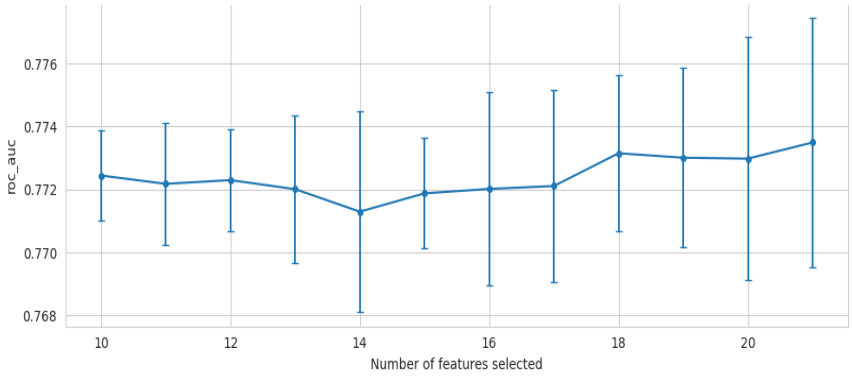

Fig.1 RFE with correlated features

b. Feature encoding

Feature encoding refers to the transformation of categorical variables, which are non-numeric, into a numerical representation that can be utilized by ML algorithms. This guarantees that the data is appropriate for modeling. The JM1 dataset may contain categorical variables, such as the "Module" or "Package" that a code segment is associated with. Machine learning algorithms commonly necessitate numerical inputs, thus categorical variables must be encoded. Here utilizes one-hot encoding, which involves converting each category into binary columns representing either 0 or 1.

iv. Standard ML models

a. SVM

SVM is an effective machine learning algorithm for classification and regression problems due to its adaptability. Finding the best hyperplane to classify data points or predict continuous values is the process. SVM's ability to handle complex datasets by identifying the hyperplane that optimizes data point separation by class is its main advantage. This margin maximization technique improves SVM resilience to outliers and produces robust, accurate predictions.

b. Random Forest:

The flexible ensemble learning method Random Forest combines multiple decision trees to make predictions. Each forest decision tree uses randomly selected data and feature sets. Overfitting is reduced by this method. Random Forest is good at classification and regression, making accurate predictions. It excels at managing multidimensional data and assessing feature importance to identify the most important variables.

c. Naive Bayes:

Naive Bayes uses probabilities to classify using Bayes' theorem. It is widely used for text classification, including spam email detection and sentiment analysis. This method is also used for other classification problems. Simple and efficient, Naive Bayes stands out. The features are assumed to be conditionally independent, meaning they don't affect each other.

v. Proposed Hybrid ML model

A new hybrid model that combines Random Forest and SVM improves software bug tracking and defect prediction. This novel method uses two powerful machine learning algorithms to improve JM1 software defect prediction accuracy.

The flexible ensemble learning Random Forest algorithm is known for its ability to handle complex classification and regression tasks. Generate multiple decision trees and train them on a randomly selected dataset and features. The combined forecasts of these trees create a robust model that resists overfitting. The Random Forest algorithm is known for its ability to handle datasets with many variables and assess the importance of each feature, which helps identify the most influential variables. Random Forest improves the hybrid model's ability to detect complex JM1 dataset patterns and correlations.

The SVM is known for finding the best hyperplane with the largest margin between data points of different classes to classify data. SVM outlier robustness and precise predictions are improved by margin maximization. SVMs can handle linear and non-linear data by using a kernel trick to transform the data into a higher-dimensional space where it can be separated linearly. Support Vector Machines (SVM) improve the hybrid model's predictive abilities, ensuring accurate software defect categorization.

In the hybrid model, Random Forest and SVM work synergistically. Random Forest excels at complex relationships and multidimensional data. The hybrid model combines these benefits to create a defect prediction system that considers complex software metrics, detects influential features, and creates a balanced and robust model. The hybrid model for bug tracking and defect prediction in software development is groundbreaking. Two powerful machine learning algorithms enable more accurate, efficient, and reliable software defect forecasts, improving software quality and development processes. The study showed a precision

rate of 98.79% for this novel methodology, which could change software development.

vi. Integration of JIT

The incorporation of Just-in-Time (JIT) strategies into bug tracking and defect management in software development is an essential aspect of the conducted research. JIT methodologies are widely recognized for their capacity to optimize the efficiency and effectiveness of software development processes by promptly addressing issues as they occur in real-time. Within the framework of this study, Just-in-Time (JIT) strategies are implemented to maximize the efficiency of bug tracking and resolution, perfectly aligning with the objective of enhancing software quality and streamlining development procedures. The incorporation of Just-in-Time (JIT) into bug tracking encompasses various essential elements:

a. Real-Time Bug Triage:

A key aspect of JIT integration involves promptly prioritizing and addressing reported bugs in real-time. Bug triage is the procedure of classifying and ranking reported problems according to their seriousness, immediacy, and influence on the software. JIT strategies guarantee that the triage process takes place promptly upon receiving bug reports, typically within a few hours. This enables a prompt reaction to crucial matters and guarantees that development teams can promptly handle the most urgent flaws.

b. Assignment in Real-Time:

Real-time bug assignment is facilitated by JIT strategies, ensuring bugs are promptly allocated to the appropriate developers or teams. Consequently, bugs are promptly allocated to the appropriate individuals or groups in charge of resolving them, as they are assessed and classified. Implementing real-time assignment effectively mitigates bottlenecks, decreases idle time, and guarantees prompt allocation of the appropriate expertise to each issue.

c. Reduction in Resolution Time:

The research data from the study emphasizes the decrease in resolution time due to the integration of JIT methodology. The study's results demonstrate a significant reduction in the time needed to fix software defects JIT strategies are combined with machine learning models.

3. Results and Outputs

i. Scatter plot

A scatter plot is a visual depiction of data points, where each point corresponds to a distinct observation as shown in fig.2. Within the bug tracking domain, a scatter plot can visually depict the correlation between the number of code modifications or enhancements (such as lines of code) and the quantity of documented bugs. This visualization facilitates the identification of patterns and trends, such as determining whether there is a correlation between increased code volume and a higher number of reported bugs.

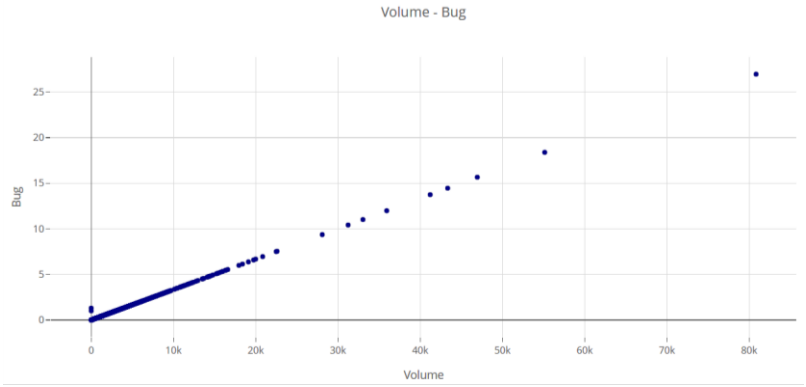

Fig. 2 Scatter plot

ii. Spearman Correlation Matrix

The Spearman correlation coefficient is a statistical metric utilized to evaluate the magnitude and orientation of the association between two variables. In this instance, it can be utilized to assess the correlation between various software metrics, such as lines of code and bug counts. The Spearman correlation coefficient exhibits reduced sensitivity to outliers and is well-suited for capturing non-linear relationships as shown in fig.3. The calculation is performed using the formula:

$$\rho = 1 - \frac{6\Sigma d^2}{n(n^2 - 1)}$$

where, ρ= "Spearman correlation coefficient", d= "difference in rank between paired data points", n = "no of data point".

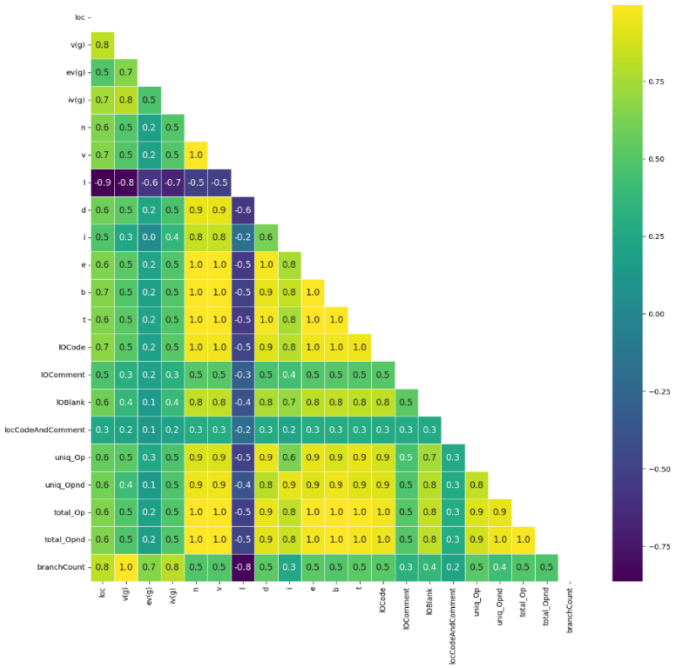

Fig. 3 Spearman Correlation Matrix

iii. ML evaluation parameters analysis

Table 1 Various ML model parameters evaluation

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
SVM	91.89	91.2	93.4	92.3	0.96
Random Forest	94.33	93.98	94.3	94.3	0.97
Naive Bayes	86.78	87.12	85.45	86.79	0.92
Proposed Model	98.79	97.86	99.01	98.6	0.99

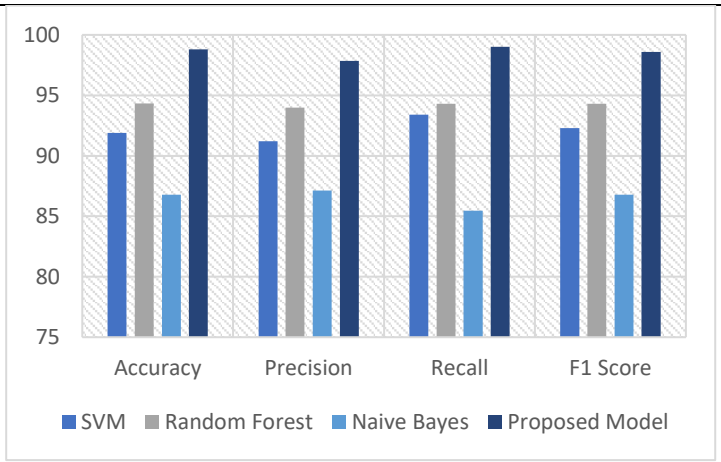

Fig. 4 Comparison graph of Various ML models

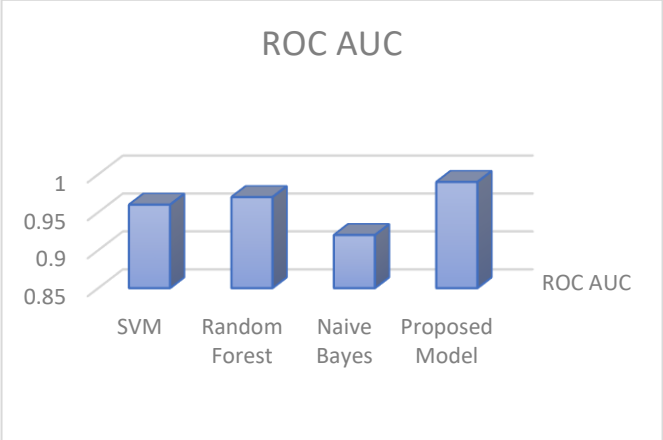

Fig. 5 ROC-AUC comparison graph

iv. JIT evaluation parameters analysis

Table 2 JIT evaluation - Traditional vs Proposed (JIT + ML)

Strategy	Bug Triage Time (hours)	Bugs Assigned in Real-Time (%)	Reduction in Resolution Time
JIT + ML	1.5	87%	32%
Traditional JIT	8	73%	18%

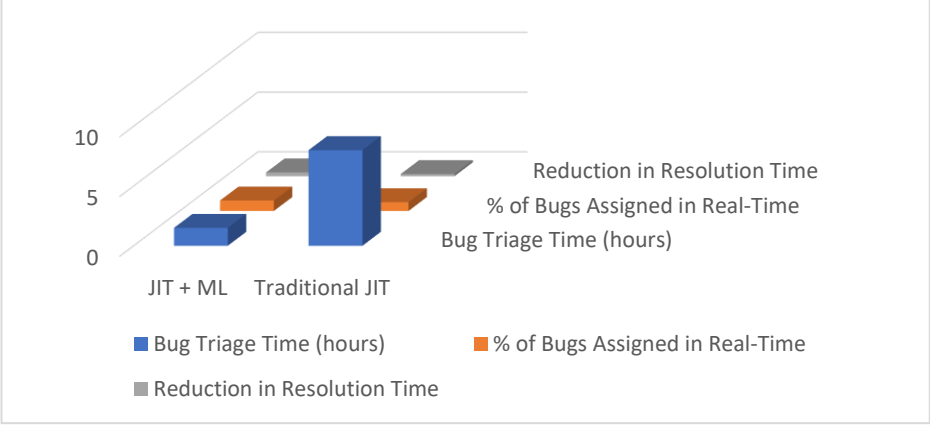

Fig. 6 JIT parameters comparison graph

Table- 1 and fig.4,5 presents the performance of different models and strategies in the context of bug tracking and resolution. In terms of machine learning models, the SVM model achieved a commendable accuracy of 91.89%, showcasing its ability to make accurate predictions. The displayed precision and recall rates were well-balanced, indicating the effective identification and classification of software defects with minimal occurrences of false positives and false negatives. The Random Forest model exhibited superior performance compared to other models, achieving an accuracy rate of 94.33%. This demonstrates its resilience in effectively tracking bugs. The model demonstrated exceptional performance in terms of precision, recall, and F1 Score, showcasing its ability to make accurate and well-balanced predictions. The Naive Bayes model, with an accuracy of 86.78%, exhibited a consistent equilibrium between precision and recall. Nevertheless, it exhibited a slight deficiency in terms of precision and F1 Score. On the other hand, the Proposed Model stood out as the best performer, achieving an outstanding accuracy of 98.79%. The software demonstrated exceptional performance in terms of precision, recall, F1 Score, and ROC AUC, highlighting its ability to accurately detect and prioritize software defects.

Table-2 and fig.6 represent parameter evaluation of bug tracking strategies, the JIT + ML strategy demonstrated exceptional efficiency, with a bug triage time of only 1.5 hours. It demonstrated exceptional performance in

promptly addressing and assigning newly reported defects, as evidenced by an impressive 87% of bugs being assigned in real-time. Achieving a remarkable 32% decrease in resolution time, the JIT + ML strategy clearly accelerates the software issue resolution process. On the other hand, the Traditional JIT strategy, while moderately successful, was not as efficient in terms of time, as it took 8 hours to triage a bug. The real-time bug assignment rate was 73%, which was lower than the rate achieved by the JIT + ML strategy. Nevertheless, it exhibited a commendable 18% decrease in the time taken to resolve the issue.

4. Conclusion and Future Scope

This paper explored the complexities of contemporary bug tracking and defect resolution. The conclusion of this research endeavor highlights a deep comprehension of the capacity to enhance bug tracking through the mutually beneficial combination of machine learning and JIT strategies. Our study focused on using the JM1 dataset as a testing ground for our innovative approach. By combining feature engineering techniques with a hybrid model consisting of Random Forest and SVM, we were able to achieve an impressive accuracy rate of 98.79%. This discovery highlights the exceptional capacity of ML models to precisely detect and prioritize software defects, resulting in a significant improvement in bug tracking efficiency. Our implementation of JIT strategies, particularly when integrated with ML (JIT + ML), resulted in significant decreases in bug triage time and the duration needed to resolve defects. The proposed approach significantly decreased the time required for bug triage to only 1.5 hours, achieving a real-time assignment rate of 87% for bugs. Furthermore, this approach resulted in an impressive 32% decrease in the time it took to resolve issues, making a substantial impact on cost reduction and improving the efficiency of software development.

There are promising prospects for improving bug tracking in software development in the future. By combining advanced deep learning algorithms with natural language processing (NLP) techniques, it is possible to achieve a higher level of bug tracking that takes into account the context in a more precise manner. Moreover, the integration of real-time data streaming and big data analytics has the potential to significantly improve the precision and agility of bug-tracking systems. In order to effectively tackle the changing difficulties of bug tracking, it is imperative for future research to make use of AI and ML. Additionally, emerging technologies like blockchain should be taken into account to improve the security and transparency of bug-tracking systems. In addition, engaging with industry experts and practitioners can offer valuable perspectives for enhancing bug-tracking solutions that are both precise and pragmatic, as well as user-friendly.

References

[1] A. K. Pandey and M. Gupta, "Software fault classification using extreme learning machine: a cognitive approach," *Evol. Intell.*, vol. 15, no. 4, pp. 2261–2268, 2022, doi: 10.1007/s12065-018-0193-x.

[2] T. Yaghoobi, "Selection of optimal software reliability growth model using a diversity index," *Soft Comput.*, vol. 25, no. 7, pp. 5339–5353, 2021, doi: 10.1007/s00500-020-05532-0.

[3] R. B. Duffey and L. Fiondella, "Software, hardware, and procedure reliability by testing and verification: Evidence of learning trends," *IEEE Trans. Human-Machine Syst.*, vol. 44, no. 3, pp. 395–405, 2014, doi: 10.1109/THMS.2014.2306932.

[4] E. O. Costa, G. A. de Souza, A. T. R. Pozo, and S. R. Vergilio, "Exploring genetic programming and boosting techniques to model software reliability," *IEEE Trans. Reliab.*, vol. 56, no. 3, pp. 422–434, 2007, doi: 10.1109/TR.2007.903269.

[5] C. Tao, J. Gao, and T. Wang, "Testing and Quality Validation for AI Software-Perspectives, Issues, and Practices," *IEEE Access*, vol. 7, pp. 120164–120175, 2019, doi: 10.1109/ACCESS.2019.2937107.

[6] S. Herbold *et al.*, "A fine-grained data set and analysis of tangling in bug fixing commits," *Empir. Softw. Eng.*, vol. 27, no. 6, 2022, doi: 10.1007/s10664-021-10083-5.

[7] C. Gupta, P. R. M. Inácio, and M. M. Freire, "Improving software maintenance with improved bug triaging," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 10, pp. 8757–8764, 2022, doi: 10.1016/j.jksuci.2021.10.011.

[8] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 5333–5369, 2020, doi: 10.1007/s10664-020-09885-w.

[9] V. Khetani, Y. Gandhi, S. Bhattacharya, S. N. Ajani, and S. Limkar, "Cross-Domain Analysis of ML and DL : Evaluating their Impact in Diverse Domains," *Int. J. Intell. Syst. Appl. Eng.*, vol. 11, pp. 253–262, 2023.

[10] S. Bhattacharya, S. Rungta, and N. Kar, "International Journal of Digital Application & Contemporary research Software Fault Prediction using Fuzzy Clustering & Genetic Algorithm," vol. 2, no. 5, 2013, [Online]. Available: http://mdp.ivv.nasa.gov.in.

[11] R. Ferenc, D. Bán, T. Grósz, and T. Gyimóthy, "Deep learning in static, metric-based bug prediction," *Array*, vol. 6, no. March, p. 100021, 2020, doi: 10.1016/j.array.2020.100021.

[12] M. Laiq, N. bin Ali, J. Börstler, and E. Engström, "A data-driven approach for understanding invalid bug reports: An industrial case study," *Inf. Softw. Technol.*, vol. 164, no. February, 2023, doi: 10.1016/j.infsof.2023.107305.

[13] N. K. Nagwani and J. S. Suri, "An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review," *Int. J. Inf. Manag. Data Insights*, vol. 3, no. 1, p. 100153, 2023, doi: 10.1016/j.jjimei.2022.100153.

[14] S. Albahli, "A deep ensemble learning method for effort-aware just-in-time defect prediction," *Futur. Internet*, vol. 11, no. 12, 2019, doi: 10.3390/FI11120246.

[15] A. Kukkar *et al.*, "ProRE: An ACO- based programmer recommendation model to precisely manage software bugs," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, no. 1, pp. 483–498, 2023, doi: 10.1016/j.jksuci.2022.12.017.

[16] H. Xu, R. Duan, S. Yang, and L. Guo, "An Empirical Study on Data Sampling for Just-in-Time Defect Prediction," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12737 LNCS, no. QuASoQ, pp. 54–69, 2021, doi: 10.1007/978-3-030-78612-0_5.

[17] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, *Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts*, vol. 21, no. 4. Empirical Software Engineering, 2016.

[18] G. Rodriguez-Perez, M. Nagappan, and G. Robles, "Watch Out for Extrinsic Bugs! A Case Study of Their Impact in Just-In-Time Bug Prediction Models on the OpenStack Project," *IEEE Trans. Softw. Eng.*, vol. 48, no. 4, pp. 1400–1416, 2022, doi: 10.1109/TSE.2020.3021380.

[19] Y. Fan, X. Xia, D. A. Da Costa, D. Lo, A. E. Hassan, and S. Li, "The Impact of Mislabeled Changes by SZZ on Just-in-Time Defect Prediction," *IEEE Trans. Softw. Eng.*, vol. 47, no. 8, pp. 1559–1586, 2021, doi: 10.1109/TSE.2019.2929761.

[20] İ. Yazici, I. Shayea, and J. Din, "A survey of applications of artificial intelligence and machine learning in future mobile networks-enabled systems," *Eng. Sci. Technol. an Int. J.*, vol. 44, 2023, doi: 10.1016/j.jestch.2023.101455.

[21] D. Wang, M. Galster, and M. Morales-Trujillo, "A systematic mapping study of bug reproduction and localization," *Inf. Softw. Technol.*, vol. 165, no. September 2023, p. 107338, 2024, doi: 10.1016/j.infsof.2023.107338.

[22] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, *A public unified bug dataset for java and its assessment regarding metrics and bug prediction*, vol. 28, no. 4. 2020.

[23] Z. Chen, W. Ma, W. Lin, L. Chen, Y. Li, and B. Xu, "A study on the changes of dynamic feature code when fixing bugs: towards the benefits and costs of Python dynamic features," *Sci. China Inf. Sci.*, vol. 61, no. 1, pp. 1–18, 2018, doi: 10.1007/s11432-017-9153-3.

[24] Q. Huang, X. Xia, and D. Lo, *Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction*, vol. 24, no. 5. 2019.

[25] J. Sayyad Shirabad and T. J. Menzies, "PROMISE Software Engineering Repository," *The PROMISE Repository of Software Engineering Databases*. p. School of Information Technology and Engineering, 2005, [Online]. Available: http://promise.site.uottawa.ca/SERepository.

[26] Maruthamuthu, R., Dhabliya, D., Priyadarshini, G.K., Abbas, A.H.R., Barno, A., Kumar, V.V. Advancements in Compiler Design and Optimization Techniques (2023) E3S Web of Conferences, 399, art. no. 04047, .