Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

Comparison Between Machine Learning Based Models and Traditional Fault Prediction Approaches for Upgrading Software Reliability

Dinesh Kumar[1], Dr. Saurabh Charaya[2]

[1]Research Scholar, Department of Computer Science and Applications,
Om Sterling Global University, Hisar, India
[2]Professor, School of Engineering and Technology,
Om Sterling Global University, Hisar, India

Abstract

Maintaining high industry standards is heavily dependent on software stability, which in turn affects product quality, customer happiness, and operational efficiency. Although they have their uses, traditional failure prediction methods aren't always up to snuff when it comes to today's complex and ever-changing software systems. In this study, Conventional research work related to fault prediction and machine learning has been discussed. Organizations can reduce maintenance costs, minimize downtime, and improve overall software quality by proactively addressing reliability issues through the integration of this paradigm into the software development lifecycle. Compared to more conventional methods of fault prediction, the ML-based approach significantly outperforms them in terms of prediction accuracy, flexibility, and scalability, according to empirical assessments. By laying out a solid plan for improving software reliability, this study advances the field and establishes new standards for the business.

Keywords: Software Reliability, Fault Prediction, Machine Learning, Industry Standards, Software Quality

1. Introduction

Ensuring software system stability is critical in today's fast-paced software development environment to meet user expectations and keep industry standards high [1]. Despite their usefulness, traditional fault prediction methods frequently fail to handle the intricacies and ever-changing nature of contemporary software, resulting in incorrect forecasts and heightened maintenance requirements [2, 3]. A more sophisticated and accurate method of defect prediction is desperately needed due to the increasing complexity of software systems [4]. An attractive alternative is machine learning (ML), which can examine massive datasets, identify complex patterns, and make better predictions of possible software errors. In order to improve software dependability, this work presents an ML-based model that outperforms conventional methods of failure prediction [5]. Reduced system downtime, decreased

maintenance costs, and a new industry standard for software quality are all goals of the proposed approach, which seeks to increase the accuracy of defect forecasts by utilizing the strengths of machine learning [6].

1.1 Background

The software industry has long struggled with the problem of guaranteeing program reliability, since even small errors can cause huge problems with operations, money, and reputation. Anticipating and reducing software errors has traditionally been accomplished through the use of methods like static code analysis, expert-based rule systems, and historical fault data analysis [7]. But contemporary software systems are notoriously difficult to adjust to, what with their huge codebases, frequent upgrades, and varied operating conditions. These more conventional methods' shortcomings have brought attention to the necessity for cutting-edge ways that can manage the complex patterns and massive datasets common in modern software development [8]. Machine learning's (ML) capacity to adapt and get better with experience has made it a hot commodity as a technique to increase the accuracy of defect predictions [9]. There is hope for ML-based models as an alternative to traditional defect prediction methods; these models have the ability to analyze large software datasets and find patterns that previous methods might miss, therefore increasing software reliability [10]. This movement towards defect prediction powered by ML is reflective of a larger industry trend towards data-driven approaches to solve complex problems and improve software quality [11].

1.2 Software Reliability

Software reliability, which measures the probability of a system functioning without failure for a set duration under specific circumstances, is an essential component of software quality. As a result, software engineers place a premium on it because of the impact it has on user confidence, operational efficiency, and the bottom line [12]. In today's fast-paced and immensely competitive technology world, dependable software is crucial because it guarantees continuous performance, prevents unexpected disruptions, and decreases the need for frequent maintenance [13]. Historically, techniques such as static code analysis, expert opinion, and historical data analysis have been used to achieve high software reliability by detecting and fixing possible defects early in the development process. Traditional methods of defect prediction in software systems typically fail to keep up with the ever-changing and interdependent nature of modern software systems [14]. Machine learning and other advanced prediction algorithms can sift through mountains of data, spot minute trends, and produce faster, more accurate fault forecasts, all of which are necessary for improving software reliability. Businesses may raise the bar for software reliability and performance by incorporating machine learning into defect prediction processes, which in turn improves software quality and performance.

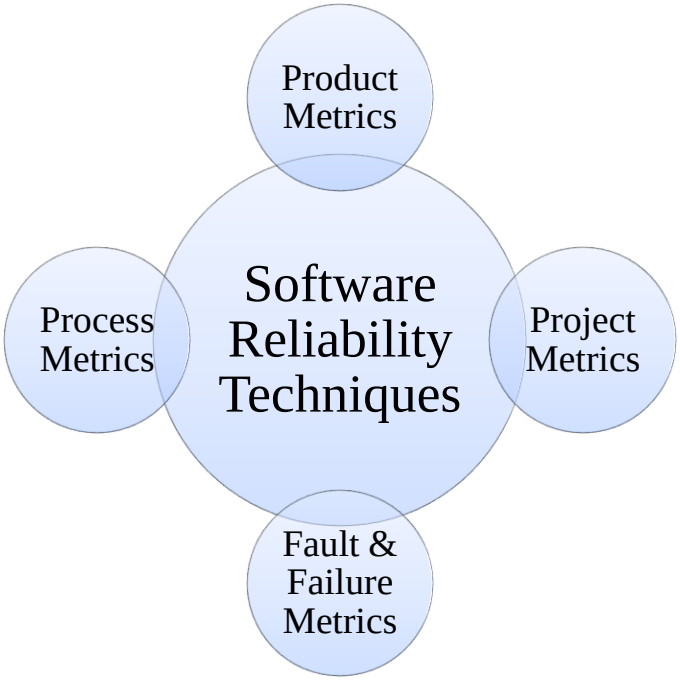

Fig 1 Software Reliability Measurement Techniques

1.3 Machine Learning

Machine learning (ML) is a game-changing technology that has revolutionized software engineering. It provides robust tools to improve software development in many ways, including reliability [15]. Machine learning algorithms improve their performance over time by learning from large and complicated datasets, in contrast to traditional methods that depend on predetermined rules and past data [16]. Code metrics, historical bug reports, and real-time system data are just a few of the numerous sources of information that ML can analyze to forecast possible software failures. Better and faster predictions are possible because these algorithms can find connections and patterns that humans or older fault prediction models can overlook. And because of their intrinsic flexibility, ML models may easily react to different software environments, codebases, and error types. Given the rapidity with which software systems are changed and deployed in modern development cycles, this adaptability is vital. Incorporating machine learning (ML) fault prediction models into software development lifecycle allows organizations to proactively handle foreseeable issues, improve software reliability, and raise industry standards for software performance and quality [17].

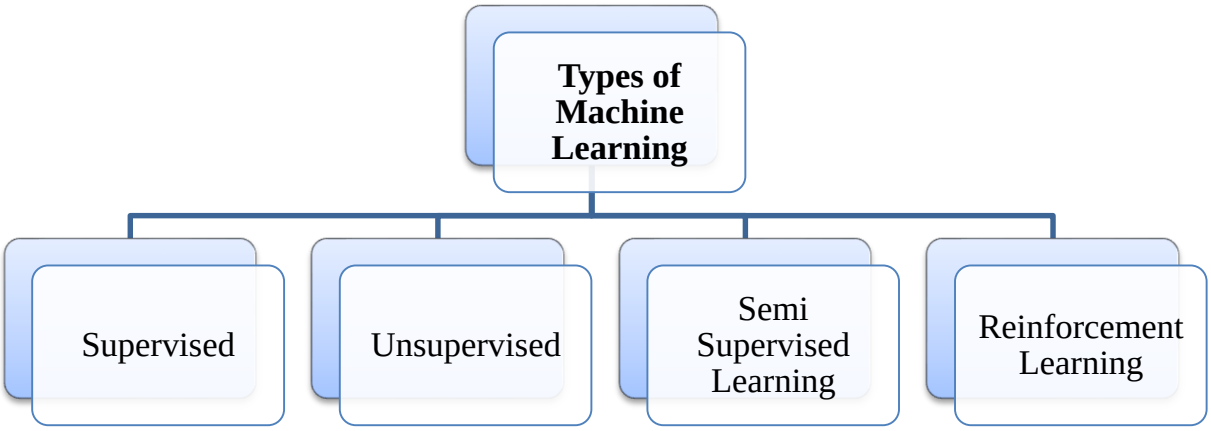

Fig 2 Types of Machine Learning

1.4 Traditional Fault Prediction

Software dependability has traditionally relied on traditional fault prediction methods, which use approaches like static code analysis, expert-based rule systems, and historical fault data investigation, to a certain extent [18]. Many methods exist for predicting the probability of software errors, but the most common ones include searching source code for patterns of known defects, using rules established from previous experiences, or making use of statistical models. The complexity and dynamism of today's software systems are outpacing the effectiveness of these solutions, which have proved useful to a certain degree [19]. Since software codebases are dynamic and run in a variety of situations, traditional methods often fall short since they rely on static rules and historical data. On top of that, these methods aren't always accurate, particularly when faced with defects that weren't there in earlier datasets. Therefore, high software reliability criteria in today's rapidly evolving technological landscape may be unattainable with the help of conventional fault prediction approaches due to their lack of accuracy and flexibility [20]. The need to find better ways to improve software reliability has prompted research into more sophisticated approaches like machine learning, which can make better failure predictions [21].

1.5 Machine Learning for Software Reliability

Machine learning (ML) has emerged as a critical tool for improving software dependability, surpassing the shortcomings of conventional fault prediction techniques [22]. Machine learning models have the ability to learn from vast and varied datasets in real-time, revealing intricate patterns and correlations that humans could overlook. This is in contrast to traditional methods that frequently depend on static rules and past data. When it comes to software reliability, ML is used to provide better defect predictions by examining a variety of inputs like code metrics, bug history, and real-time performance measurements [23]. These models excel at learning from new data to enhance their predictions over time, adapting to dynamic software environments. With the use of ML, enterprises may improve the reliability of their software systems by detecting possible issues earlier in the software development lifecycle, reducing the occurrence of undetected faults, and eliminating them altogether. Because of this,

software becomes more reliable, which in turn reduces maintenance costs, increases user satisfaction, and enhances software quality and robustness, all of which help to raise industry standards.

1.6 Significance of research

Efficacious machine learning-based models have the ability to improve software dependability, which is important since it might solve the problems with current defect prediction methods and establish new benchmarks for the industry. Modern software is complicated and constantly evolving, making traditional solutions that depend on static rules and past data ineffective [24]. An effective substitute is machine learning, which uses sophisticated algorithms to sift through varied and expansive datasets, spot intricate patterns, and adjust to new program settings. In turn, this skill improves software reliability and decreases maintenance costs by allowing for more accurate and timely defect predictions. Machine learning models that improve defect detection and work in tandem with current software development methods to provide predictions and insights in real time are the focus of this study. More resilient systems, better software quality, and happier users are the results of this study's advancements in defect prediction, which in turn lead to better industry standards. With a more effective answer to the problems that old methods have, this research is going to revolutionize software dependability management.

1.7 Motivation of research

The increasing requirement for very dependable software in sectors where even small mistakes can have major ramifications, along with the increasing complexity of contemporary software systems, is driving this research. Despite its worth, traditional defect prediction methods are falling short in handling the complexities of modern software environments. As a result, there are more and more dependability gaps and maintenance expenses are going up. More sophisticated, precise, and adaptable fault prediction systems are urgently required since software is integral to every aspect of modern life, from corporate processes to consumer goods [25]. To tackle these issues, this study investigates how machine learning (ML) might improve software dependability. Improved software quality, fewer system failures, and higher industry standards can be achieved by using ML's capacity to handle massive amounts of data and reveal hidden patterns to create a defect prediction model. The end goal is to deliver a solid foundation that can handle the present and future needs of software reliability while also establishing a new standard for software engineering.

2. Literature review

A new set of software metrics, Erme-type software metrics (ESM), was introduced by P. Khoa et al. in 2023. ESM gives prediction models information on patterns of various Java runtime errors. The most common lav runtime issues, including Index Out of Bounds Exception, Null Pointer Exception, and Class Cast Exception, are detailed in the ESM values. The proposed Error-type software metrics did, in fact, greatly enhance the accuracy with which machine learning models predicted the likelihood of errors. [1]

In their 2023 article, G. Youdi et al. present a thorough analysis of dataset quality. This presents a thorough methodology for evaluating quality, including a framework for evaluating datasets using

dimensions and metrics, as well as methods for computing quality measures and assessment models. The overall advancements in evaluating dataset quality are emphasized. [2]

A. Elmishali et al. built failure prediction models in 2023 using software measurements as features. They introduced a new paradigm for building features called Issue-Driven features, which integrate software metrics with requirements information. An analysis of 86 open-source projects reveals that, compared to state-of-the-art features, Issue-Driven features outperform them by 6% to 13% in terms of AUC. [3]

Ensemble Machine Learning handles feature irrelevance, missing data, and uneven distribution between defective and now faulty classes. T. Sharma et al. demonstrated this in 2023. The performance of defect prediction is also shown to have been improved. The writers have made an effort to understand the patterns, techniques, and dataset that academics use to find software problems. For the purpose of software defect prediction, this article examines all ensemble-based machine learning methods that have been created between 2018 and 2021. [4]

S. Pandey et al. (2023) reviewed state-of-the-art algorithms for software failure prediction, with an emphasis on problems with class imbalance. In order to help the researcher choose the most effective methods for software failure prediction, we provide a comparative presentation of several datasets and algorithms. Further observation: SMOTE is the data sampling technique most often employed to address data quality issues. [5]

Made from samples of the majority and bound with samples of the minority, these are called Manually Disjoint Data Sets (MDS). Ensemble models were generated by applying a collection of diverse machine learning methods to the bound nut-samples. When using a double-voting procedure, ensembles of mutually disjoint data sets prioritize minority samples. On average, this strategy improves recall by 13.72 percent on test data, compared to just 3 percent on train data. [6]

In 2023, the effectiveness of various machine learning algorithms in identifying malware on Android was examined by A. Hani et al. To get the highest level of accuracy, it utilizes PCA, normalizes the numerical features, and the Synthetic Minority Over-sampling Technique (SMOTE). To detect and categorize families of android malware, a light GBM model is suggested. Based on the results, the Light GBM model outperforms the other techniques that were tested in terms of accuracy. Light GBM achieves an F-1 score of 95.47 percent. [7]

Software defect prediction using deep learning algorithms was the goal of Batool et al. in 2023. The trials employ RBFN, BILSTM, and LSTM, three deep learning algorithms. In terms of accuracy, the LSTM algorithm achieves 93.53% and the BILSTM algorithm 93.75%, respectively, which is greater performance. Nonetheless, RBFN achieves an accuracy of 82.58%. While LSTM and BILSTM are both fast algorithms, RBFN outperforms them by a significant margin. [8]

An effective method for predicting software failures using hybrid machine learning techniques was presented by R. Chennappan et al. in 2023. The first step is to optimize the dataset's features using a Genetic Algorithm (GA), which allows for feature selection with a better fitness function. Once the best characteristics have been chosen, a classification approach called the Decision Tree (DT) algorithm is employed to process them. [9]

S. D. Immaculate and colleagues (2019) In order to improve speed, dependability, and quality, software engineers are increasingly turning to Machine Learning methods for bug prediction. This method constructs models and makes bug predictions from past data using Logistic regression, Naïve Bayes, and Decision Tree classifiers. The models are designed to operate effectively in all settings using random forest ensemble classifiers and K-Fold cross validation [10].

Software fault prediction (SFP) is a method developed by Caulo, M. et al. (2019) that employs software metrics to generate predictive models. These models are based on machine learning and statistical approaches. For metrics to be organized and communicated consistently, a taxonomy is required. A global grasp of metrics—acronyms, full names, descriptions, and research articles—is being sought for, and this doctoral symposium paper details current efforts to build such a taxonomy [11].

A new method for improving malware detection systems using a feature hashing methodology is presented by Moon et al. (2022). By studying how to decrease the dimensionality of feature sets while maintaining detection accuracy, they hope to make machine learning-based malware detection more efficient. Significant gains in computational efficiency and accuracy are demonstrated by the suggested method, which could have far-reaching ramifications for cybersecurity.[12]

Y. Liu, et. al. (2022) conduct a comparative study to understand the impact of data imbalance on software defect prediction. Their findings highlight how imbalanced datasets can affect the performance of defect prediction models, stressing the importance of addressing this issue to enhance the accuracy and reliability of the predictions. They suggest potential strategies for mitigating this challenge in software defect prediction tasks [13].

N. Sharma, et al. (2021) explores various applications of machine learning and deep learning in different domains. The authors present a vision for the future of these technologies, emphasizing their growing significance in various fields such as healthcare, finance, and software engineering. They argue that machine learning and deep learning models can greatly improve decision-making and automation capabilities across diverse sectors [14].

A. El-Kilany, et al. (2021) propose a novel adversarial-guided oversampling technique (TGT) to tackle the issue of imbalanced datasets in machine learning tasks, specifically in the context of software defect prediction. This technique aims to generate synthetic data points that help balance the class distribution, thereby improving the overall performance of machine learning models in detecting software defects [15].

M. Mangla, (2021) introduce a sequential ensemble model designed to improve software fault prediction. By combining multiple models in a sequence, the ensemble approach aims to reduce errors and enhance prediction accuracy. This work contributes to the field by providing a more robust method for predicting software defects using machine learning techniques [16].

B. Mumtaz, et al. (2021) focuses on feature selection using an artificial immune network approach for software defect prediction. The authors argue that selecting the right features is crucial to enhancing the performance of defect prediction models. Their work demonstrates the effectiveness of artificial immune networks in selecting relevant features from large and complex datasets [17].

M. Mustaqeem (2021) presents a hybrid technique for software defect detection that combines Principal Component Analysis (PCA) with Support Vector Machine (SVM). The authors claim that the integration of PCA for dimensionality reduction and SVM for classification results in a more accurate and efficient software defect detection model, reducing the complexity of the prediction task [18].

A. Rahim, et al. (2021) explore the use of the Naive Bayes classifier for software defect prediction. They evaluate the effectiveness of this simple yet powerful machine learning algorithm in predicting defects, comparing it with other more complex models. Their results indicate that Naive Bayes can be a viable option for software defect prediction, particularly when dealing with large datasets [19].

M. Nevendra (2021) examine of deep learning techniques for software defect prediction. The authors argue that deep learning, with its ability to automatically extract features from raw data, holds significant promise for improving the accuracy of defect prediction models. Their findings suggest that deep learning models outperform traditional machine learning techniques in terms of prediction accuracy [20].

E. M. Rey, (2021) integrated iterative machine teaching and active learning into the machine learning loop. The authors propose that these techniques can be used to enhance the learning process by actively selecting the most informative data points and iterating over them to improve model performance. Their approach aims to make machine learning systems more efficient and accurate by reducing the number of required labeled samples [21].

S. K. Rath, (2022) present a comparative analysis of reliability prediction models in software engineering. They compare various methods for predicting software reliability, focusing on their accuracy and applicability to different types of software systems. Their study contributes to the field by providing insights into the strengths and weaknesses of different reliability prediction techniques [22].

A. S. Mohamad (2022) introduces a machine learning-empowered softwion system that integrates Support Vector Machine (SVM) and Extreme Learning Machine (ELM) classification techniques. The study compares the performance of these models for predicting software defects, demonstrating the potential of machine learning techniques in improving defect prediction accuracy [23].

K. Sofian, et al. (2022) proposes the Salp Swarm Optimizer (SSO) to model the software fault prediction problem. The authors demonstrate the effectiveness of this metaheuristic optimization algorithm in improving the accuracy of software fault prediction models, suggesting that SSO can be a valuable tool for tackling complex prediction tasks [24].

D. Mohammad et al. (2022) discusses the development of a machine learning-powered software defect prediction system, focusing on the integration of intelligent automation techniques to enhance the prediction accuracy. The authors propose a system that leverages machine learning models to predict software defects more efficiently and accurately, integrating various automation methods to streamline the process [25]. The field of software defect prediction has seen significant advancements through the application of various machine learning and optimization techniques. Research efforts have been directed toward understanding and improving the reliability of software by predicting fault-prone modules. Choudhary et al. (2018) [26] conducted an empirical analysis of change metrics and

highlighted their effectiveness in software fault prediction, emphasizing the importance of dynamic change data over static attributes. Similarly, Kaur (2018) [27] evaluated multiple classification algorithms, providing insights into their applicability for fault prediction in open-source projects.

Manjula and Florence (2018) [28] introduced a hybrid machine learning approach combined with optimization techniques, demonstrating improved prediction accuracy and reduced false positives. Hybrid methodologies were further explored by Rhmann (2018b) [29], (2018a) [30], who applied hybrid search-based algorithms for cross-project defect prediction, addressing the challenges of dataset diversity and feature variance. Sharma and Chandra (2018) [31] presented a comparative analysis of soft computing techniques, underscoring their relevance in developing efficient fault prediction models.

The role of machine learning techniques, such as those outlined by Raschka and Mirjalili (2017) [32], has been pivotal in advancing defect prediction frameworks. Malhotra (2017) [32] proposed a framework for defect prediction in Android software using empirical approaches, while Yang et al. (2015) [33] employed a learning-to-rank methodology for prioritizing defect-prone modules, enhancing prediction reliability. Studies by Erturk and Sezer (2015) [34] and Zhou et al. (2010) [38] have compared complexity metrics and soft computing methods, revealing their varying efficacy across different project types.

Foundational works, including those by Moser et al. (2008) [39] and Kim et al. (2008) [40], have examined the efficiency of change metrics and static code attributes, respectively, providing benchmarks for subsequent research. Additionally, the application of fuzzy rule-based classifiers (2006, 2004) [42, 43] has been explored as a means to handle uncertainty in defect prediction tasks, showcasing the evolution of methodologies from deterministic to probabilistic approaches.

These studies collectively highlight the growing sophistication in software defect prediction, with a shift toward hybrid and optimization-driven approaches to enhance model performance and applicability in diverse software development environments.

Table 1 Literature survey

Ref	Author / Year	Objective	Technique	Limitation
[1]	Khoa et al. (2023)	Propose a novel set of software metrics for fault prediction	Error-Type Metrics	May not be applicable to all software types
[2]	Youdi et al. (2023)	Survey on dataset quality in ML	Literature Review	Limited to quality aspects only
[3]	Elmishali et al. (2023)	Investigate issue-driven features for fault prediction	Issue-Driven Features	Focuses on specific issues, may not generalize

[4]	Sharma et al. (2023)	Explore ensemble ML paradigms in defect prediction	Ensemble Methods	May have high computational costs
[5]	Pandey et al. (2023)	Survey recent developments in fault prediction for imbalanced data	Survey of Techniques	Focuses on imbalance handling only
[6]	Koyyada et al. (2023)	Multi-stage approach for class imbalance	Ensemble Method	Complexity in model implementation
[7]	Hani et al. (2023)	Comparative analysis of ML for malware detection	Comparative Analysis	Limited to Android malware
[8]	Batool et al. (2023)	Software fault prediction using deep learning techniques	Deep Learning	May require extensive computational resources
[9]	Chennappan et al. (2023)	Automated software failure prediction using hybrid ML	Hybrid ML Algorithms	Hybrid methods may increase model complexity
[10]	Immaculate et al. (2019)	Unsupervised machine learning for software defect prediction	Supervised Learning	May not account for all types of software bugs
[11]	Caulo et al. (2019)	Taxonomy of metrics for software fault prediction	Taxonomy Analysis	May not cover all existing metrics
[12]	Moon et al. (2022)	Compact feature hashing for malware detection	Feature Hashing	May affect detection accuracy for some malware
[13]	Y. Liu, (2022)	Study effect of data imbalance on software defect prediction	Comparative study of imbalanced datasets	Focuses only on imbalance, doesn't address other factors
[14]	N. Sharma, (2021)	Explore machine learning and deep learning applications	Overview of ML and DL applications	Limited to applications, lacks specific implementation
[15]	A. Mahmoud, (2021)	Propose a novel oversampling technique for imbalanced datasets in defect prediction	Adversarial guided oversampling (TGT)	Oversampling might introduce noise in certain contexts

[16]	M. Mangla, (2021)	Introduce an ensemble model for software fault prediction	Sequential ensemble learning	Ensemble approach can lead to increased computational complexity
[17]	B. Mumtaz, (2021)	Use artificial immune networks for feature selection in defect prediction	Artificial immune network for feature selection	May not scale well with very large datasets
[18]	M. Mustaqeem, (2021)	Combine PCA and SVM for software defect detection	PCA + SVM	PCA may lose critical information due to dimensionality reduction
[19]	A. Rahim, (2021)	Use Naive Bayes classifier for software defect prediction	Naive Bayes classifier	Simple classifier might not capture complex patterns
[20]	M. Nevendra, (2021)	Apply deep learning for software defect prediction	Deep learning techniques	DL require large datasets and high computational resources
[21]	E. M. Rey, (2021)	Integrate iterative machine teaching and active learning into ML loops	Machine teaching + Active learning	Active learning may not always find the most informative data points
[22]	S. K. Rath, (2022)	Perform a comparative analysis of software reliability prediction models	Various software reliability models	Doesn't provide a comprehensive approach to feature selection
[23]	A. S. Mohamad (2022)	Develop an SVM and ELM-based prediction system for software defects	SVM + ELM	May face challenges with noisy data or outliers
[24]	K. Sofian, (2022)	Apply SSO for fault prediction	SSO	Optimization approach might struggle with local minima
[25]	D. Mohammad et al. (2022)	Develop a machine learning-driven software defect prediction system	Machine learning and intelligent automation	High computational costs for integrating automation
[26]	Choudhary et	Empirical analysis of change metrics for software	Change metrics	Limited focus on generalizability across

	al., 2018	fault prediction		varied datasets
[27]	Kaur & Kaur, 2018	Evaluate classification algorithms for fault prediction in open-source projects	Classification algorithms	Lack of exploration into ensemble methods
[28]	Manjula & Florence, 2018	Develop a hybrid approach for software defect prediction using machine learning with optimization	Hybrid machine learning and optimization	Computational complexity and scalability issues
[29]	Rhmann, 2018b	Cross-project defect prediction using hybrid search-based algorithms	Hybrid search-based algorithms	Limited applicability to large-scale datasets
[30]	Rhmann, 2018a	Application of hybrid search-based algorithms for software defect prediction	Hybrid search-based algorithms	Performance variability depending on the dataset
[31]	Sharma & Chandra, 2018	Comparative analysis of soft computing techniques in fault prediction model development	Soft computing techniques	Limited insight into real-world project scenarios
[32]	Raschka & Mirjalili, 2017	Develop frameworks for machine learning-based defect prediction	Python-based machine learning approaches	Narrow focus on selected machine learning algorithms
[33]	Yang et al., 2015	Learning-to-rank approach to software defect prediction	Learning-to-rank	Focus limited to ranking rather than classification
[34]	Erturk & Sezer, 2015	Compare soft computing methods for fault prediction	Soft computing methods	No consideration of hybrid or ensemble approaches
[35]	Fenton & Bieman, 2015	Provide a rigorous approach to software metrics	Software metrics	Lack of focus on predictive techniques
[36]	Kaur & Kaur, 2014	Evaluate machine learning algorithms for fault prediction	Machine learning algorithms	Limited scope of evaluation metrics
[37]	Nam et al., 2013	Transfer defect learning for software fault prediction	Transfer learning	Limited applicability across dissimilar

				datasets
[38]	Zhou et al., 2010	Assess the predictive capability of complexity metrics in object-oriented systems	Complexity metrics	Limited evaluation of real-time scenarios
[39]	Moser et al., 2008	Compare change metrics and static code attributes for defect prediction	Change metrics and static code attributes	Lack of integration with advanced machine learning techniques
[40]	Kim et al., 2008	Classify software changes as clean or buggy	Classification-based approach	No emphasis on improving fault detection accuracy
[41]	Gondra, 2008	Apply machine learning for software fault-proneness prediction	Machine learning	Limited generalizability across varied software projects
[42]	Otero & Sanchez, 2006	Develop fuzzy classifiers using the Logitboost algorithm	Fuzzy classifiers with Logitboost	Computational intensity and limited interpretability
[43]	Jesus et al., 2004	Induction of fuzzy rule-based classifiers with evolutionary boosting algorithms	Fuzzy rule-based classifiers with evolutionary boosting algorithms	Scalability and adaptability challenges

2.1 Research Gap

Considering existing research work the research gap is discussed below

1. **Limited Adaptability:** Traditional fault prediction approaches often rely on predefined rules or heuristics that may not adequately adapt to the dynamic nature of modern software systems. There is a gap in the literature regarding the adaptability of these approaches to evolving software environments and changing user requirements.
2. **Scalability Challenges:** Conventional fault prediction methods may struggle to handle large-scale software systems with numerous components and dependencies. There is a gap in understanding how traditional approaches can scale effectively to address the complexities of modern software architectures.
3. **Lack of Predictive Accuracy:** While traditional fault prediction methods have been widely used, their predictive accuracy may be limited, particularly in identifying subtle patterns or complex relationships between software metrics and fault occurrences. There is a gap in research exploring novel techniques to improve the accuracy of fault prediction models.

4. **Data-driven Insights:** Traditional fault prediction approaches often rely on manual expertise or historical data analysis, which may overlook valuable insights hidden within large datasets. There is a gap in understanding how machine learning techniques can harness data-driven insights to enhance fault prediction accuracy and effectiveness.
5. **Interpretability and Explainability:** Machine learning models, particularly complex ones, may lack interpretability and explainability, making it difficult for stakeholders to trust and understand the predictions. There is a gap in research exploring techniques to improve the interpretability and explainability of machine learning-based fault prediction models.
6. **Generalization Across Domains:** Machine learning models trained on specific software datasets may struggle to generalize across different domains or application contexts. There is a gap in understanding how to build machine learning models that can generalize effectively to diverse software systems and environments.

The field of fault prediction faces several gaps, including the generalization and applicability of metrics and techniques. Khoa et al. introduced a set of software metrics in case of fault prediction, but these may not be universally applicable across all software types. Caulo (2019) provided a taxonomy of metrics in case of fault prediction, but it may not cover all existing metrics or their effectiveness in diverse contexts. Youdi et al. (2023) conducted a survey on dataset quality in ML but there is a need to understand how different dimensions of dataset quality impact fault prediction models. Sharma et al. (2023) and Hani et al. (2023) examined ensemble and hybrid machine learning methods for defect prediction and malware detection, but they often come with high computational costs and increased complexity. Addressing these gaps could significantly advance the field by developing more adaptable metrics, improving dataset quality and imbalance, along with refining ensemble and hybrid methods for broader and more practical use.

2.2 Challenges

The software fault prediction field faces several challenges, including applicability of metrics across different software and environments, the need for adaptable metrics, the complexity of dataset quality and class imbalance, the need for deeper investigation into how different aspects impact fault prediction, the high computational costs and complexity of ensemble and hybrid methods, and the need to bridge the gap between theoretical advancements and real-world application. These challenges require continued research and development to create versatile, efficient, and practical solutions for software fault prediction along with related fields. By addressing these challenges, the field can continue to advance and improve accuracy along with efficiency of software fault prediction.

3. Issues or Problem Statement

Software reliability is a significant challenge in modern software development due to increasing complexity and scale of software systems. Traditional fault prediction methods, such as static code analysis and rule-based systems, are limited by their reliance on historical data and predefined rules, which may not adapt well to evolving software structures or new fault types. This results in incomplete or inaccurate fault predictions, increasing the risk of software failures, maintenance costs, and system downtimes. The need for more effective and adaptive solutions is evident as software systems continue

to grow in complexity and demand. Advanced methods like machine learning are needed to improve overall software reliability.

4. Research Methodology

Improving software reliability with machine learning is an area of study that follows a well-defined methodology with multiple important steps. Among these steps are gathering and preparing data, doing EDA engineering features, selecting a model, training it, evaluating it, doing comparison analysis, and finally, deploying it. The first stage involves gathering diverse datasets from various sources, ensuring their quality through rigorous preprocessing. The second stage involves analyzing the cleaned data to uncover patterns, relationships, and anomalies. The third stage involves feature engineering, selecting and creating informative features to enhance model performance. The fourth stage involves selecting the appropriate machine learning models based on problem requirements, data characteristics, and computational complexity. The fifth stage involves model training, dividing the dataset into training and testing sets, and optimizing model. The sixth stage involves model evaluation, comparing the model's performance with traditional methods. The final stage involves deploying the validated model into operational environments, integrating it into existing software development workflows or fault management systems.

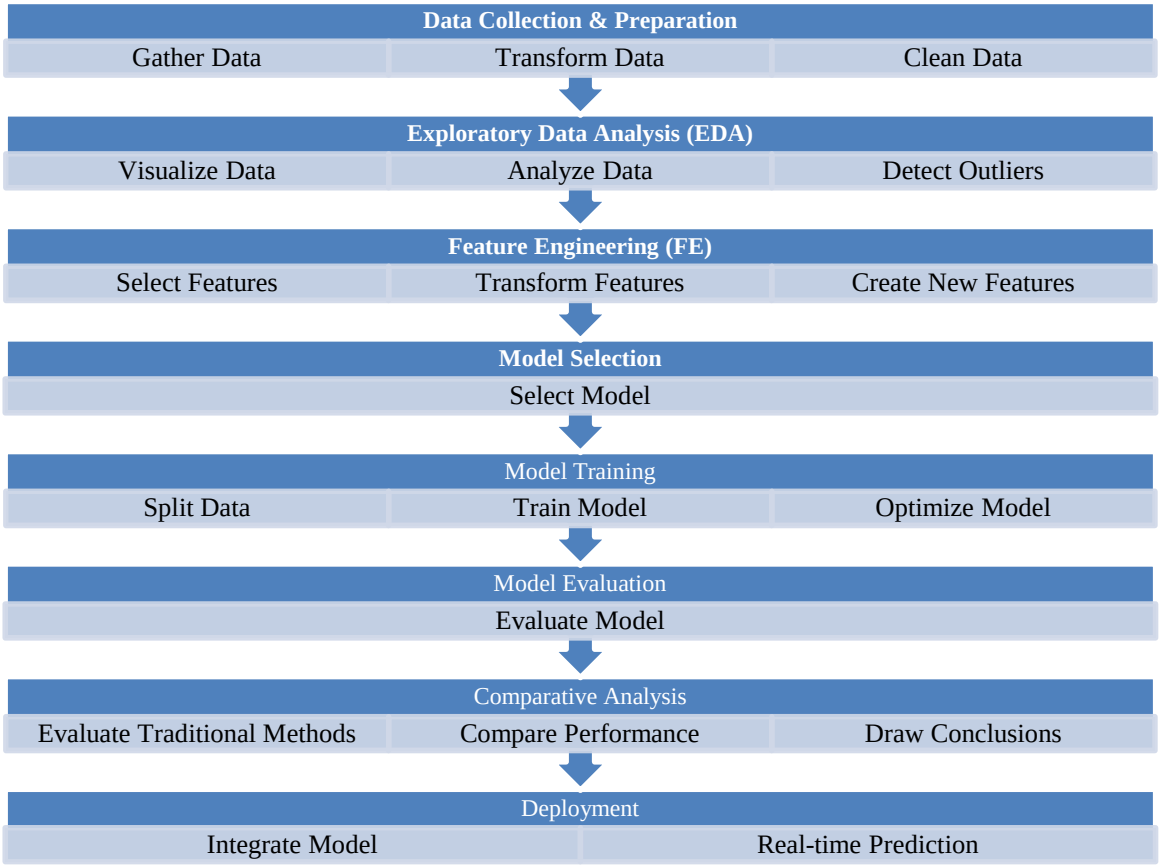

Fig 3 Research methodology

5. Need of Research

The complexity and criticality of modern software systems necessitate research into enhancing software reliability through machine learning. Traditional fault prediction approaches, based on static rules and historical data, are inadequate for accurately predicting faults and ensuring software reliability. Machine learning offers advanced analytical capabilities to process large data volumes, identify complex patterns, and provide more accurate fault predictions. Research in this area is crucial for developing effective machine learning-based models that address limitations, improve fault detection accuracy, and enhance software reliability. This advancement can set new industry standards, reduce downtime, and increase software system robustness, benefiting users and organizations.

6. Conclusion

Machine learning is poised to revolutionize software reliability by enhancing models that can handle diverse and dynamic data sources. As software systems grow, there's a need for sophisticated models that can adapt to new fault types and software environments. Integrating machine learning with emerging technologies like edge computing and blockchain could offer real-time fault detection and prevention. Deep learning techniques, transfer learning, and automated machine learning could improve model performance and reduce manual tuning. Research into interpretability and transparency of machine learning models is crucial for trust and understanding decision-making processes. By advancing these areas, future research can drive significant improvements in software reliability, set industry standards, and contribute to more resilient and dependable software systems.

References

1. P. Khoa, O. Emmanuel and A. Mehmet (2023, March), "Error-Type-A Novel Set of Software Metrics for Software Fault Prediction," IEEE Access, vol. 11, pp. 30562-30574.
2. G. Youdi, L. Guangzhen, X. Yanzhi, L. Rui and M. Lingzhong (2023, June), "A survey on dataset quality in Machine Learning." Information and Software Technology, vol. 162, pp. 1-11.
3. A. Elmishali and M. Kalech (2023), "Issue-Driven Features for Software Fault Prediction," Informatica and Software Technology, vol. 155, pp.1-8.
4. T. Sharma, A. Jatain, S. Bhaskar and K. Pabreja (2023), "Ensemble Machine Learning Paradigms in Software Defect Prediction," Procedia Computer Science, vol. 218, pp. 199-209.
5. S. Pandey and K. Kumar (2023), "Software Fault Prediction for Imbalance Data: A Survey on Recent Developments," Procedia Computer Science, vol. 218, pp. 1815-1824.
6. S. P. Koyyada and T. P. Singh (2023), "A multi stage to handle class imbalance: An ensemble method," Procedia Computer Science, vol. 218, pp. 1815-2666-2674.
7. A. Hani, M. Y. Qussai and A. A. B. Mohammed (2023), "A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection," Procedia Computer Science, vol. 220, pp. 763-768.
8. Batool and T. A. Khan (2023), "Software Fault Prediction Using Deep Learning Techniques," Software Quality Journal, pp. 1-16.

9. R. Chennappan and V. (2023), "An automated software failure prediction technique using hybrid machine learning algorithms," Journal of Engineering Research, vol. 11, pp. 1-5
10. Immaculate, S. D., Begam, M. F., & Floramary, M. (2019, March). “Software bug prediction using supervised machine learning algorithms”, In 2019 International conference on data science and communication (IconDSC) ,pp. 1-7. IEEE.
11. Caulo, M. (2019, August). A taxonomy of metrics for software fault prediction. In Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (pp. 1144-1147).
12. D. Moon, 1. Lee and M. Yoon (2022), "Compact festure hashing for machine learning based malware detection," ICT Express, vol. 8, pp. 124-129.
13. Y. Liu, W. Zhang, G. Qin and 1. Zhao (2022), "A comparative study on the effect of data imbalance on software defect prediction," Procedia Computer Science, vol. 214, pp. 1606-1616.
14. N. Sharma, R. Sharma and N. Jindal (2021, January), "Machine Learning and Deep Learning Applications-A Vision", Global Transitions Proceedings, vol. 2, pp. 24-28.
15. A. Mahmoud, A. El-Kilany, F. Ali and S. Mazen (2021, February), "TGT: A Novel Adversarial Guided Oversampling Technique for Handling Imbalanced Datasets", Egyptian Informatics Journal, vol. 22, pp. 433-438.
16. M. Mangla, N. Sharma, and S. N. Mohanty (2021, March), "A sequential ensemble model for software fault prediction," Innovations Systems Software Engineering, vol. 18, pp. 301-308.
17. B. Mumtaz, S. Kanwal, S. Alamri and F. Khan (2021, April), "Feature Selection Using Artificial Immune Network: An Approach for Software Defect Prediction," btelligent Automation & Soft Computing, vol. 29, pp. 669-684.
18. M. Mustaqeem and M. Saqib (2021, April), "Principal component-based support vector machine (PC- SVM): a hybrid technique for software defect detection," Cluster Computing, vol. 24, pp. 2581-2595.
19. A. Rahim, Z. Hayat, A. Rahim, M. A. Rahim and M. Abbas (2021, November), "Software Defect Prediction with Nalve Bayes Classifier," 2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST), Islamabad, Pakistan, pp. 293-297.
20. M. Nevendra and P. Singh (2021, November), "Software Defect Prediction using Deep Learning." Acte Polytechnica Hungarica, vol. 18, no. 10, pp. 173-189.
21. E. M. Rey, D. A. Rios and A. B. Lozano (2021), "Integrating Iterative Machine Teaching and Active Learning into the Machine Learning Loop," Procedia Computer Science, vol. 192, pp. 553-562
22. S. K. Rath, M. Sahu, S. K. Bisoy, S. P. Das and M. Sain (2022, August), "A Comparative Analysis of reliability Prediction Model," Electronics, vol. 11.
23. SVM and ELM Classification on Software [28] A. S. Mohamad (2022, October), "Machine Learning Empowered Software Prediction System," Wani Journal of Computer and Mathematic Science, vol 1, no. 3, pp. 54-64.
24. K. Sofian, A. Salwani, A. A. B. Mohammed and A. Mohammed (2022), "Salp swarm optimizer for modeling the software fault prediction problem," Journal of King Saud University-Computer and Information Sciences, vol. 34, pp. 3365-3378.
25. D. Mohammad et al. (2022), "Machine Learning Empowered Software Defect Prediction System Activate Windows Intelligent Automation & Soft Computing, vol. 31, pp. 1287-1300.

26. Choudhary, G.R., Kumar, S., Kumar, K., Mishra, A., Catal, C., 2018. Empirical analysis of change metrics for software fault prediction. Comput. Electr. Eng., Elsevier 67, pp. 15–24.
27. Kaur, A., Kaur, I., 2018. An empirical evaluation of classification algorithms for fault prediction in open source projects. J. King Saud Univ.-Comput. Inf. Sci., Elsevier 30, pp. 2–17.
28. Manjula, C., Florence, L., 2018. Hybrid approach for software defect prediction using machine learning with optimization technique. Int. J. Comput. Inf. Eng., World Acad. Sci. Eng. Technol. 12 (1), pp. 28–32.
29. Rhmann, W., 2018b. Cross project defect prediction using hybrid search based algorithms. Int. J. Inf. Technol., Springer,. https://doi.org/10.1007/s41870-018-0244-7.
30. Rhmann, W., 2018a. Application of hybrid search based algorithms for software defect prediction. Int. J. Modern Educ. Comput. Sci., MECS 10 (4),pp. 51–62. https://doi.org/10.5815/ijmecs.2018.04.07.
31. Sharma, D., Chandra, P., 2018. A comparative analysis of soft computing techniques in software fault prediction model development. Int. J. Inf. Technol., Springer pp. 1–10. https://doi.org/10.1007/s41870-018-0211-3.
32. Raschka, S., Mirjalili, V., 2017. Python Machine Learning. Published by Packt Publishing Ltd. Malhotra, R., 2016. An empirical framework for defect prediction using machine learning techniques with Android software. Appl. Soft Comput., Elsevier 49 (C),pp. 1034–1050.
33. Yang, X., Tang, K., Yao, X., 2015. A learning-to-rank approach to software defect prediction. IEEE Trans. Reliab. 64 (1), pp. 234–246.
34. Erturk, E., Sezer, E.A., 2015. A comparison of some soft computing methods for software fault prediction. Expert Syst. Appl., Elsevier 42 (4), pp. 1872–1879.
35. Fenton, N., Bieman, J., 2015. Software Metrics. A Rigorous and Practical Approach. CRC Press, Taylor and Francis group.
36. Kaur, A., Kaur, I., 2014. Empirical evaluation of machine learning algorithms for fault prediction. LNSE Lecture Notes Software Eng. 2 (2), pp. 176–180.
37. Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. Proc. of Int'l Conf. on Softw.Eng. (ICSE'13), pp. 382–391.
38. Zhou, Y., Xu, B., Leung, H., 2010. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. J. Syst. Softw. 83, pp. 660–674.
39. Moser, R., Succi, Pedrycz W., 2008. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction May 10–18. ICSE'08, Leipzig, Germany, pp. 181–190.
40. Kim, S., Whitehead, E.J., Zhang, Y., 2008. Classifying software changes: clean or buggy. IEEE Trans. Softw. Eng. 4 (2),pp. 181–196.
41. Gondra, I., 2008. Applying machine learning to software fault-proneness prediction. J. Syst. Softw. 81, pp. 186–195.
42. Otero, J., Sanchez, L., 2006. Induction of descriptive fuzzy classifiers with the Logitboost algorithm. Soft Comput., Springer 10, pp. 825–835.
43. Jesus, M.J., Hoffmann, F., Navascues, L.J., Sunchez, L., 2004. Induction of fuzzy-rulebased classifiers with evolutionary boosting algorithms. IEEE Trans. Fuzzy Syst. 12 (3), pp. 296–308.