Software Fault Detection using ML/DL Algorithms

21BCE037 Chaudhari Rushali 21bce037@nirmauni.ac.in
21BCE192 Eva Patel 21bce192@nirmauni.ac.in
21BCE246 Neha Rana 21bce246@nirmauni.ac.in

Institute Of Technology, Nirma University
Department Of Computer Science and Engineering

***Abstract*—This review of software fault prediction using ML and DL algorithms/techniques contrasts classic ML techniques with DL models, such as RBFN, BiLSTM, and LSTM, to see which is more successful at predicting software defects early in the SDLC. The assessment looks at indications, parameter sensitivity, and mobile app scenarios using data from several projects. Methods such as feature engineering and data balancing are investigated to improve performance, especially for LSTM-based models. The outcomes highlight deep learning's potential for accurate software defect prediction. SVMs are good at clearly separating classes in the ML and DL domain, but Random Forests and Decision Trees provide transparent decision-making. The study uses metrics like accuracy, area under the curve, and Cohen's kappa to compare software failure prediction efficiency and stresses a balanced approach between interpretability and model complexity.**

***Index Terms*—RBFN, BILSTM, LSTM, SDLC, RNN, Data Mining, ML, DL, SVM, ANN, CNN, Decision Trees, Random Forest, KNN, Logistic Regression, Gradient Boosting, Autoencoders, Preprocessing, Performance metrics**

I. Introduction

Software fault prediction (SFP) seeks to increase our understanding of the predictive capacity of deep learning methods for software faults, emphasizing the significance of early error detection in the software development life cycle. By comparing the results with common machine learning techniques, the study evaluates the prediction power of RBFN, BILSTM, and LSTM for software defects. The case study's goal is to provide insightful information on parameter variation analysis's application to deep learning model optimization. Cross-project data is used to anticipate software defects in the context of mobile apps, where the analysis is also expanded. Through the exploration of feature engineering and data balancing strategies, special attention is made to LSTM-based architectures as potential performance enhancers for deep learning models.

The findings offer crucial information about the practical application of deep learning methods for software failure prediction, offering insights into their efficacy and potential impacts on software development processes. Given that there are various ML algorithms with varying benefits and drawbacks, an examination of DL and ML techniques is required for software failure prediction. Decision trees and random forests offer transparency, which is essential in circumstances when an understanding of the decision-making procedures is needed. When there is a clear separation of classes, SVMs perform best. Deep learning adds complexity and flexibility, which is highly helpful for finding intricate patterns in datasets including software flaws. Three categories of neural networks: ANNs, CNNs, and RNNs—have the potential to simulate intricate interactions and connections observed in sequential and grid-structured data.

Additionally, autoencoders and gradient boosting techniques which offer unique benefits for the fault prediction industry are included in the research. This thorough investigation is crucial to the development of predictive modeling because it provides practitioners with empirical data that aids in their decision-making regarding the best approaches to employ in order to predict software problems. The following sections provide in depth evaluations of both ML and DL algorithms, together with performance and comparison considerations, problems, and limitations pertaining to their application in software failure prediction. In software engineering, the ability to predict and prevent software issues is essential. This work performs an extensive evaluation of software problem prediction using datasets from NASA programs, academic research initiatives, and commercial software engineering methodologies.

The focus also includes evaluation of machine learning techniques including Naive Bayes, Rotation Forest, and SVM. Furthermore, the study explores the potential of recurrent and convolutional neural networks, concentrating on models like LSTM and BiLSTM, as well as the realm of deep learning. The primary objective is to improve the software's predictive capacity, with particular attention to accuracy and reliability. The rigorous three-step methodology employed in this work begins with preprocessing and dataset selection, continues with the application of several ML algorithms, and concludes with DL techniques.

The effectiveness of the models is assessed using a range of performance assessment metrics, which also shed light on how well the models relate to the intricate subject of software failure prediction.

II. Related Work

A. SDP using DL Techniques

DL algorithms for SDP, including RBFN, BILSTM, and LSTM, are examined in this case study. This[1] is summarised as follows: The goal of the case study is to discover errors early in the SDLC by applying DL techniques for SFP .The study highlights the potential for DL to yield more accurate findings when comparing the efficacy of ML and DL techniques for software defect prediction.[2]The paper analyses SDP using DL algorithms including RBFN, BILSTM, and LSTM. It investigates how these DL approaches' parameters may be changed and how their performance is measured. The case study also looks at the use of DL techniques to detect software errors in mobile applications by utilising cross-project data.[3]The case study also explores how feature engineering and data balance strategies might enhance the performance of DL models. It also discusses how much DL architectures—in particular, models based on LSTMs—can forecast software errors in mobile apps .The ability of DL techniques to increase the accuracy of early software development fault diagnosis and to anticipate software faults is demonstrated by these studies. This case study offers insightful information on the real-world implementation of DL methods for SFP,[4] providing a thorough grasp of their efficacy and possible influence on software development procedures.

B. RNN-based DL method and ensemble ML techniques for SFP

The application of an RNN-based DL method and ensemble ML techniques for software fault prediction is examined in the case study [5] "SFP Using an RNN-Based DL Approach and Ensemble ML Techniques." [6] In comparison to other ML and DL algorithms, the study shows how accurate the RNN-based method is in predicting software faults. The study looks at how well ensemble ML approaches and a DL approach based on RNNs predict software errors. It highlights how crucial it is to precisely pinpoint errors early in the SDLC in order to improve the quality and dependability of software.The paper highlights the benefits of utilising RNN-based models for SDP by comparing the performance of the RNN-based method with other ML and DL techniques. It sheds light on the experimental findings and shows how much more accurate the RNN-based method is when compared to other methods. The case study's conclusions advance knowledge of the possible applications of ensemble ML and RNN-based DL methods for software defect prediction. The goal of the research is to offer useful insights for utilising DL and advanced ML techniques to improve software quality and forecast accuracy of faults. This case study provides a thorough examination of the use of ensemble ML and RNN-based DL approaches for SDP including insightful information about their efficacy and possible effects on software development processes.[7] [8] [9] [10]

C. SFP Using DL Algorithms

[11] In this case study, we look at using DL techniques to forecast software faults. "SFP Using DL Algorithms,"[11] with a focus on early defect construct identification in the development lifecycle. By identifying which parts of a big software system are most likely to have the most defects in the upcoming release, the research seeks to increase software quality and dependability. The study examines the predictive power of DL algorithms such as CNN, RNN, and LSTM. It investigates how these DL approaches' parameters may be changed and how their performance is measured. Furthermore, the case study explores the use of feature engineering and data balancing strategies to enhance the performance of DL models. Finally, the case study looks into the use of cross-project data for mobile applications to predict software faults using DL approaches. This case study [12] provides an in-depth examination of the use of DL algorithms for software fault prediction, offering insightful information about their efficacy and potential effects on software defects. It also discusses the degree to which DL architectures, specifically LSTM-based models, can forecast software faults for mobile applications. The study's findings advance knowledge of the potential of DL techniques in software fault prediction and their capacity to improve the accuracy of fault identification in the early stages of SD.[7] [13]

D. SFP Using Data Mining, ML and DL Techniques

[14] The case study [14] "SFP Using Data Mining, ML and DL Techniques: A Systematic Literature Review" examines how well support vector machines, decision trees, and random forests perform as ML methods for software fault prediction. The paper gives a thorough overview of the methods utilised in this field and focuses on identifying ML and data mining approaches used for SFP. [2] In addition to outlining the benefits and problems in utilising these approaches to find software defects early in the development lifecycle, [15]the research looks into how well various ML algorithms forecast software flaws. The case study sheds light on the use of ML methods to improve fault tolerance, such as decision trees, random forests, and support vector machines. It highlights how these methods may enhance fault prediction and automatically capture the semantic representation of programmes. In addition, the research methodically detects, evaluates, condenses, and synthesises the status of the use of ML algorithms for software defect prediction, providing insightful information on the difficulties and developments in this field.The performance of many ML approaches for SDP is thoroughly examined in this case study, which reveals insightful information about their efficacy and possible effects on SD procedures.[16] [17]

E. SFP Using ML Techniques

The use of ML techniques for SFP is examined in the case study [18]"SFP Using ML Techniques". In an effort to attain quality, maintainability, and reusability, it places a strong emphasis on the use of decision trees, artificial neural networks, and support vector machines to find flaws, mistakes, ambiguities, and foul odours in software. [19] Furthermore, the ”Progress on ML Techniques for SFP” case study offers information about the advancements achieved in the use of ML techniques for SFP. It talks about how important fault prediction is to SE and how to find components that are prone to errors. Furthermore included in the case study ”Software Fault Prediction Using DL Techniques” is the application of ML methods for SFP , such as artificial neural networks, decision trees, support vector machines, and evolutionary algorithms. It highlights how critical it is to find errors early in the SD life cycle and contrasts these approaches' effectiveness with DL techniques.Additionally, a thorough overview of the application of ML techniques for SFPis given by the systematic literature review ”SFP using data mining, ML, and DL techniques”. It places a strong emphasis on the early detection of errors in the SD life cycle and the promise of ML-based techniques for fault prediction. These case studies and literature analysis provide insightful information about the use of ML methods for software failure prediction, such as support vector machines, artificial neural networks, and decision trees. They offer a thorough comprehension of the difficulties, developments, and possible effects of ML techniques in SFP.[18] [20]

III. Analysis of the ML/DL Algorithms

Within software failure prediction, examining ML and DL methods is a vital task with numerous ramifications. The reason behind this investigation is the variety of machine learning algorithms; each has distinct advantages and disadvantages that need for a careful analysis to determine whether or not they can be used in defect prediction scenarios.[21] Decision Trees and Random Forests provide transparent, comprehensible models when a thorough grasp of decision-making procedures is necessary.[22] Support Vector Machines (SVM) are a reliable option for some defect prediction scenarios and perform well when there is clear class separation. Simultaneously, the introduction of deep learning (DL) adds a degree of complexity and flexibility that is especially useful for identifying complicated patterns in software defect datasets. [23]

Three types of neural networks: ANNs, CNNs, and RNNs all show promise in modeling intricate connections and relationships found in sequential and grid-structured data.[23] The investigation of Autoencoders and Gradient Boosting techniques is also included in the analysis since each offers a distinct benefit to the field of software fault prediction. This thorough analysis is essential for refining prediction skills, adjusting algorithmic decisions to the properties of the current dataset, and striking a balance between interpretability and model complexity trade-offs.[24]

By comparing how well ML and DL algorithms work, practitioners can obtain empirical insights that help them make well-informed decisions about which approaches are best for predicting software faults. Essentially, the critical examination of ML and DL algorithms contributes to the theoretical advancement of predictive modeling as well.

A. *Analysis of the ML algorithms*

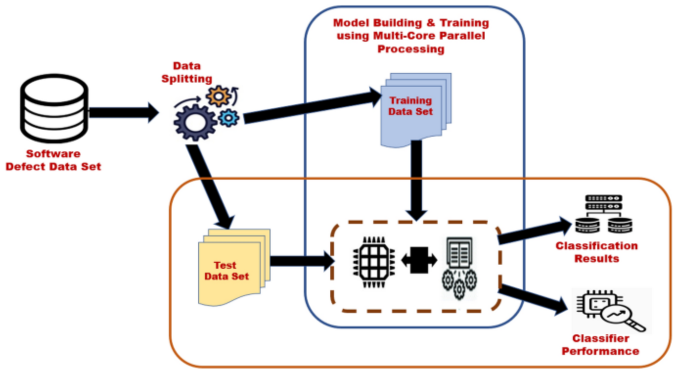
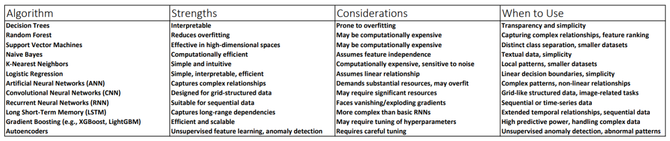


Fig. 1. software fault prediction using machine learning

1) **Decision Trees**: Decision trees can be used for activities that require transparency in the decision-making process since they are interpretable.[25] They can, however, overfit due to their sensitivity to noise and outliers. Overfitting can be reduced by pruning methods, which include reducing the depth of the tree or cutting off branches with poor prediction ability. While decision trees work well for representing simple associations, they may not be able to handle complicated patterns found in large-scale datasets.[25]
2) **Random Forest**: By creating an ensemble of trees, Random Forests reduce overfitting and increase resilience in order to overcome the constraints of decision trees. They work well at capturing the intricate correlations seen in big datasets, which makes them appropriate for predicting software failures where a variety of factors are involved.[26] The feature relevance ranking that Random Forests offers helps find important predictors. Their application is increased by their capacity to manage noisy surroundings and is more complex and has a very highly dimensional part in the data field.
3) **SVM**: SVMs work well in high-dimensional environments when there are distinct margins separating instances that are defective from those that are not. For large datasets, they could become computationally costly, though, thus it's important to choose the right kernel functions and adjust the hyperparameters. SVMs perform well in situations where there is a clear class distinction, which makes them appropriate for software failure prediction in those situations. For scalability, they need to use approximation or optimization approaches.[27]
4) **Naive Bayes**: Large-scale datasets can benefit from the computational efficiency of Naive Bayes. Its efficacy in capturing intricate interactions, however, might be constrained by its assumption of feature independence. Since Naive Bayes works well in text classification tasks, it can be applied to software failure prediction in situations where textual data is important. Naive Bayes,

for all its simplicity, is not beyond producing competitive answers, particularly when feature independence is a realistic assumption.[28]

5) **KNN**: Although KNN is simple to use and intuitive, the calculating of distances between data points makes it computationally costly for big datasets. Scalability may be increased by using effective data structures like KD-trees.[29] For software fault prediction jobs where local patterns are crucial, KNN works well when instances with comparable characteristics are likely to have similar fault labels. Crucial factors to take into account are deciding on the right distance metric and figuring out how many neighbors are ideal (k). [30]
6) **Logistic Regression**: Binary classification tasks can benefit from the interpretability, computational efficiency, and suitability of Logistic Regression. It works well in situations where the decision boundary is roughly linear since it assumes a linear relationship between the features and the log odds of the outcome. For software fault prediction assignments where a clear grasp of the link between characteristics and defects is crucial, logistic regression is a good fit. [31]



Algorithm	Strengths	Considerations	When to Use
Decision Trees	Interpretable	Prone to overfitting	Transparency and simplicity
Random Forest	Reduces overfitting	May be computationally expensive	Capturing complex relationships, feature ranking
Support Vector Machines	Effective in high-dimensional spaces	May be computationally expensive	Distinct class separation, smaller datasets
Naive Bayes	Computationally efficient	Assumes feature independence	Textual data, simplicity
K-Nearest Neighbors	Simple and intuitive	Computationally expensive, sensitive to noise	Local patterns, smaller datasets
Logistic Regression	Simple, interpretable, efficient	Assumes linear relationship	Linear decision boundaries, simplicity
Artificial Neural Networks (ANN)	Captures complex relationships	Demands substantial resources, may overfit	Complex patterns, non-linear relationships
Convolutional Neural Networks (CNN)	Designed for grid-structured data	May require significant resources	Grid-like structured data, image-related tasks
Recurrent Neural Networks (RNN)	Suitable for sequential data	Faces vanishing/exploding gradients	Sequential or time-series data
Long Short-Term Memory (LSTM)	Captures long-range dependencies	More complex than basic RNNs	Extended temporal relationships, sequential data
Gradient Boosting (e.g., XGBoost, LightGBM)	Efficient and scalable	May require tuning of hyperparameters	High predictive power, handling complex data
Autoencoders	Unsupervised feature learning, anomaly detection	Requires careful tuning	Unsupervised anomaly detection, abnormal patterns

Fig. 2. A general analysis of both the algorithms

B. Analysis of the DL algorithms

1) **ANN**: Software defect prediction challenges containing complicated patterns are a good fit for artificial neural networks (ANNs) due to their capacity to simulate complex connections. Deep architectures can cause overfitting, though, and they require a significant amount of data and processing power. Because of its ability to capture non-linear correlations well, ANNs are useful in situations when other models might not be as successful. Optimizing performance requires careful hyperparameter optimization.[32]
2) **CNN**: In order to capture hierarchical features, CNNs are best suited for grid-structured data, such as photographs. When the input material has a grid-like layout, like code snippets or pictures pertaining to the development process, they work well in software defect prediction tasks.[33] Through weight sharing, CNNs improve scalability by reducing the number of parameters. They could, however, demand a large amount of processing power, particularly for deep structures.
3) **RNN**: RNNs work well with sequential data, but they have trouble with long-range dependencies and vanishing/exploding gradients. Specialized RNN variants called LSTMs effectively capture extended temporal relationships and address these problems. [7]For software defect prediction jobs requiring time series or sequential data, RNNs and LSTMs are appropriate. The hyperparameter tweaking and architecture selection may have an impact on their performance.[5]
4) **LSTM**: LSTMs are intended to overcome the drawbacks of simple RNNs and capture long-range dependencies in sequential data. They work well for applications requiring the prediction of software faults when it is essential to comprehend temporal connections. Because LSTMs perform well in situations involving complex temporal linkages, they can be used to forecast problems based on the chronological order of past events. But they need to be carefully considered because of their intricacy and possible sensitivity to hyperparameter selections.[8]
5) **Gradient Boosting (e.g., XGBoost, LightGBM)**: Gradient Boosting techniques, like XGBoost and LightGBM, create a sequence of weak learners. They offer strong prediction capability and handle large-scale datasets with efficiency. These algorithms are adaptable and appropriate for jobs involving the prediction of software defects, where it is essential to capture intricate correlations and manage a variety of factors that impact problems. They are useful for real-world applications because of their scalability and efficient implementation.[34]
6) **Autoencoders**: As unsupervised learning models, autoencoders work well for anomaly detection and feature learning. They are appropriate for software defect prediction when anomalous pattern identification is crucial because they can reveal latent representations in the data. For best results, the architecture selected, the hyperparameters adjusted, and the ratio of anomaly detection sensitivity to reconstruction accuracy must all be balanced. Autoencoders offer versatility in capturing underlying data structures in a variety of fault prediction scenarios.[35]

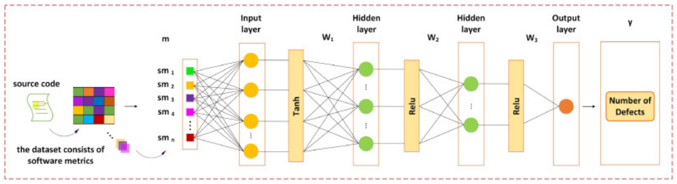


Fig. 3. Software fault prediction using deep learning

C. Comparative analysis

The main goal of this comparison investigation is to assess how well deep learning (DL) and machine learning (ML) algorithms predicted software faults. The efficacy of the algorithms in detecting and forecasting errors was evaluated using critical measures including accuracy, recall, and F1 score. Interestingly, the examination turned up different patterns in the two paradigms.

1) ML Algorithm Performance : Decision Trees and Random Forests showed excellent interpretability, bringing decision-making procedures to light. Nevertheless, the overfitting inclinations of Decision Trees offset their simplicity, a problem that Random Forests' ensemble learning strategy helped to address. Support Vector Machines (SVM) demonstrated resilience in situations where class separation was well-defined; but, computational costs were noticeable in bigger datasets. Known for its ease of use, logistic regression worked well in situations where the decision boundaries were linear, providing a performance-to-interpretability ratio.

High predictive power was demonstrated by gradient boosting techniques, such as XGBoost, which successfully captured intricate correlations in the data. To fully realize their potential, hyperparameters have to be adjusted. The Naive Bayes method showed promise in terms of computing performance, especially when it came to text data settings; nonetheless, its assumption of feature independence presented challenges.[34][21][33]

2) Cross-Algorithmic Considerations : Although machine learning methods such as SVMs and Decision Trees provided interpretability, they were unable to fully capture the complex linkages and non-linear interactions seen in the data. Conversely, fault prediction tasks including a variety of complicated data structures are a good fit for DL algorithms, especially ANNs, which showed an impressive capacity to identify subtle patterns. CNNs performed particularly well in situations where defect identification relied heavily on attributes connected to images, as they were built for grid-structured data.[33][22]

3) Model Robustness and Adaptability : Machine learning algorithms, including Random Forests and Gradient Boosting, demonstrated resilience in identifying many elements that lead to errors. Random Forests' ensemble learning technique improved prediction accuracy, whereas Gradient Boosting's boosting methodology reduced mistakes by gradually enhancing model performance. Still, the capacity of DL models—particularly LSTMs—to capture temporal relationships in sequential fault data demonstrated a clear benefit in tasks where knowledge of the temporal evolution of faults was needed.[2][36]

4) Scalability and Efficiency : Efficiency and scalability concerns dominated the analysis. Because of their computational efficiency, these techniques of machine learning are appropriate in situations where resource limitations are a major factor. However, there was a trade-off with DL models, especially ANNs, as they provided better prediction but at the cost of more computational complexity. This emphasizes the necessity of taking a balanced approach, matching algorithmic decisions to the amount of processing power at hand and the size of the task of predicting software faults.[37][34]

5) Ensemble Learning vs. Specialized Architectures: Through the merging of many models, Random Forests and Gradient Boosting demonstrated enhanced performance, highlighting the advantages of ensemble learning in machine learning techniques.[38] On the other hand, DL models used specialized designs to capture subtle dependencies in sequential fault data, such the recurrent nature of LSTMs. Comprehending these architectural differentiations offers significant perspectives for practitioners who are choosing algorithms according to the type of their fault prediction assignment.[21][20]

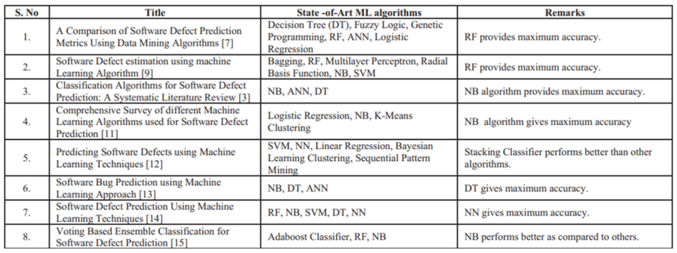
6) DL Algorithm Performance : ANN has proven to have an unmatched ability in the field of DL to reduce the complicated correlations from software defect data. Convolutional Neural Networks (CNNs) have demonstrated remarkable efficacy in processing grid-structured data, rendering them a powerful tool for defect prediction applications involving images. Long Short-Term Memory networks (LSTMs) demonstrated their effectiveness in collecting long-range dependencies in sequential data, particularly in situations where the temporal linkages were extended.[7]

7) Comparative Insights : In contrast to conventional ML techniques, the comparison study demonstrated the subtle advantages of DL algorithms, especially ANNs and LSTMs, in recognizing intricate patterns. The performance of DL algorithms was better, particularly in cases where complex dependencies and linkages were important elements in failure prediction. It is essential to recognize, nonetheless, that the efficacy of every method depends on the particular features of the dataset and the fault prediction job itself.[39]

8) Transferability Across Diverse Domains: One interesting observation was the varying degree of transferability of ML and DL models across different software domains. Random Forests and other machine learning algorithms have demonstrated some universality by consistently delivering good results across a range of datasets. Conversely, DL models—particularly ANNs—showed evidence of domain-specific adaptation, working well in certain software contexts but requiring care in others.[30]

9) Explainability and Trustworthiness: The importance of reliability factors has become apparent in safety-critical software systems, highlighting the necessity of striking a balance between interpretability and performance.

10) Unsupervised Learning for Anomaly Detection: The use of unsupervised learning, in particular autoencoders, for anomaly identification in fault prediction was an intriguing direction that was investigated. Autoencoders proved adept at uncovering latent representations of typical software system behavior, which makes them useful for spotting unusual patterns that could point to possible problems. This unsupervised method showed potential, particularly in situations with a lack of labeled fault data.



S. No	Title	State -of-Art ML algorithms	Remarks
1.	A Comparison of Software Defect Prediction Metrics Using Data Mining Algorithms [7]	Decision Tree (DT), Fuzzy Logic, Genetic Programming, RF, ANN, Logistic Regression	RF provides maximum accuracy.
2.	Software Defect estimation using machine Learning Algorithm [9]	Bagging, RF, Multilayer Perceptron, Radial Basis Function, NB, SVM	RF provides maximum accuracy.
3.	Classification Algorithms for Software Defect Prediction: A Systematic Literature Review [3]	NB, ANN, DT	NB algorithm provides maximum accuracy.
4.	Comprehensive Survey of different Machine Learning Algorithms used for Software Defect Prediction [11]	Logistic Regression, NB, K-Means Clustering	NB algorithm gives maximum accuracy
5.	Predicting Software Defects using Machine Learning Techniques [12]	SVM, NN, Linear Regression, Bayesian Learning Clustering, Sequential Pattern Mining	Stacking Classifier performs better than other algorithms.
6.	Software Bug Prediction using Machine Learning Approach [13]	NB, DT, ANN	DT gives maximum accuracy.
7.	Software Defect Prediction Using Machine Learning Techniques [14]	RF, NB, SVM, DT, NN	NN gives maximum accuracy.
8.	Voting Based Ensemble Classification for Software Defect Prediction [15]	Adaboost Classifier, RF, NB	NB performs better as compared to others.

Fig. 4. POPULAR ML ALGORITHMS USED BY RESEARCHERS IN RECENT PAST
[40]

D. Challenges and Limitations of the algorithms

There are certain difficulties and restrictions when using both of the techniques in this field is to detect software faults. One significant issue in machine learning is the possibility of overfitting, especially when using decision trees and random forests. These algorithms may overfit to the training set, identifying noise instead of true patterns, which would lower their ability to generalize results on unobserved data. Furthermore, certain machine learning models, like gradient boosting, include sophisticated ensemble techniques that can be difficult to comprehend, which can be a drawback since it makes it harder to understand how decisions are made, which is important in circumstances where software defect prediction calls for transparency.[31]

One recurring probelm in the field of DL is the need for large volumes of data. Large datasets are ideal for training convolutional and recurrent architectures in deep neural networks. The ideal performance of deep learning models may be limited by the limited availability of comprehensive and diverse fault data, which might result in overfitting on smaller datasets. Furthermore, DL models are infamous for their complexity, which presents problems with training timeframes and computer resources. Complex neural networks can be computationally demanding to train, which limits their usefulness in contexts with limited resources or for businesses with low-level computational infrastructure.[36]

The potential difficulty in managing unbalanced datasets, a prevalent situation in software fault prediction where the occurrence of problems is often a minority class, is another barrier that ML and DL share. The efficacy of defect prediction can be reduced when models emphasize precision above accurately identifying the minority class due to imbalance.

Overarching issues still exist in interpreting and explaining the judgments made by both ML and DL models. Complex ensemble models and deep neural networks frequently function as ”black boxes,” making it more complex to learn how and why a exact prediction is produced. In contrast, simpler machine learning models, such as decision trees, provide transparency. This lack of interpretability can be a serious disadvantage, especially for safety-critical software systems where it’s essential to comprehend the reasoning behind forecasts.

To sum up, the difficulties and constraints encountered by ML and DL algorithms for predicting the fault in the software include overfitting, interpretability, data needs, and computational complexity. To overcome and modify these limits for more reliable and useful applications in the software engineering domain, addressing these problems requires a deep understanding of the particularities of fault prediction jobs and ongoing research.[13][41][27]

1) Overfitting Dynamics: One notable difficulty that has surfaced is the vulnerability of machine learning techniques, especially gradient boosting, to hyperparameter adjustment. The careful tuning of hyperparameters was necessary to strike a balance between computational efficiency and model complexity, highlighting the significance of tuning for the best results.

2) Hyperparameter Sensitivity: A prominent challenge that has emerged is the susceptibility of machine learning methods, particularly gradient boosting, to hyperparameter manipulation. To balance model complexity and computing performance, rigorous hyperparameter adjustment was required, emphasizing the need of fine-tuning for optimal outcomes.[26]

3) Interpretability Trade-offs: Interpretability persisted as a major issue in the difficulties related to both ML and DL. While DL models functioned as ”black boxes,” making it difficult to understand the reasoning behind particular predictions, ML algorithms offered transparency. The practical application of these algorithms in fault prediction scenarios depends on finding a compromise between interpretability and predictive strength.

4) Ethical Considerations: Beyond technical issues, the study looked at the ethical ramifications of using ML and DL to forecast software errors. Concerns regarding justice and bias were among the possible risks, particularly with relation to the training set of data. Ethical considerations should be taken into account when creating and deploying predictive models, as it became imperative to eliminate biases and assure justice in algorithmic decision-making.[3]

5) Dynamic Nature of Software Systems: The robust nature of the software systems presented one major challenge. Because previous machine learning methods were designed for static datasets, it was challenging for them to adapt to the evolving characteristics of software failures. DL models have proven to be resilient in managing the dynamic character of faults because of their ability to describe temporal dependencies. This makes them more suitable in scenarios where fault patterns change over time.[32]

IV. Proposed model/Algorithm

This study uses many datasets from NASA programs and industry practices, among other domains, to study software problem prediction. It evaluates a number of DL models, such as CNN and RNN, in addition to ML techniques like Naive Bayes. Improving the prediction capacity of software issues is the objective. It provides a three-step method to improve forecast accuracy and uses metrics to evaluate efficiency. Improving the accuracy and reliability of software failure prediction is the main objective of the work.

A. Selection of datasets and preprocessing

In [41] fifteen publicly available datasets from the PROMISE set were used to reflect NASA projects, academic research, open-source Apache applications, and industrial software engineering practices in Turkish white-goods production. Based on the quantity of faults (higher than 0), code metrics were categorized as faulty (fp) or non-faulty (nfp), and they were subsequently converted to binary for machine learning algorithms. The SFP XP-TDD dataset was used to evaluate the predictive value of the dataset for software issues using performance indicators.

B. Machine Learning Algorithm

Using well-known machine learning algorithms author [5] describes Naive Bayes, SVM, KStar, RF, and K-Nearest to transform inputs into meaningful outputs is a popular strategy in the [5]. An ensemble classifier is a unique type of classifier which improves overall accuracy by combining multiple simple classifiers. They are frequently employed in rotation forests, random forests, bagging, and boosting.

Like Random Forest, Rotation Forest creates complex models by combining several forecasts while maintaining the essential components of ensemble techniques. Rotation Forest's distinct feature is that it chooses randomized subsets of the dataset properties for every model, which may help to ensure that a variety of attributes are employed and may reduce model bias. The outcomes demonstrated its wide application in classifier ensembles.

C. Deep Learning

Relative to simple machine learning algorithms, which usually comprise one or two layers, modern deep learning models have more layers. A model's depth is directly correlated with the number of layers in it. The number of training iterations in the deep neural network training stages is shown in [5], which are obtained by representative learning across processing layers over epochs. In the [5] on DL, CNN and RNN are considered foundational designs. These networks are necessary for image processing, bio signal processing, text classification, facial recognition, and health applications. Layer depth affects these models' capacity to express complex patterns, and they eventually pick up meaningful notations. The success of CNN and RNN in various areas can be attributed to their sequential processing and multi-layered architecture, which enable them to adapt to the ever-changing demands of deep learning applications.

D. Convolutional Neural Network (CNN)

Multi-layered ANN called CNNs, are well known for their capacity to solve visual recognition issues. In addition to image processing, CNN is flexible enough for a variety of additional uses, such as audio processing. Its distinct convolutional layer sets it apart from conventional neural networks and enables it to analyze matrix data in addition to image data. Due to the unique capabilities of its convolutional layers, which let it to distinguish objects such as edges, corners, and complex textures inside complex systems, CNN performs at image processing tasks. This flexibility demonstrates CNN's suitability in a variety of unusual picture recognition scenarios. Because of its forward-looking architecture, which helps in the resolution of complicated patterns and structures, the network is more beneficial across a range of industries. CNN's layered approach to audio processing facilitates the identification and classification of complex audio input. This application demonstrates CNN's broad applicability to data processing outside of the visual arts.

E. Recurrent Neural Network (RNN)

The Author [5] deep learning systems called RNNs are made for continuous data analysis. Values from the stage before are integrated into each loop phase's design. Its name comes from the fact that it performs an operation while accounting for past outputs for each element in the array. For this reason, it's called "repetitive." It produces better learning results than simpler neural network techniques. Through the use of recurrent processes, the RNN links recently received forms with previously received values, so establishing a connection between the present and memory-stored data.

Two modified RNN variations are called Long Short-Term Memory and Bidirectional Long Short-Term Memory. Compared to standard RNNs, LSTM networks have an advantage because they record both cell state and hidden state statistics. Cells are memory chunks used by LSTM networks. This is very helpful when working with time-series or organized data. LSTM functions well with sequential data because cells monitor values inside memory blocks.

BiLSTM, an innovation in RNN design, functions as two separate RNN structures to process inputs both forward and backward. Because the bidirectional technique allows the system to record input at any given time from both the past and the present, it enhances the system's comprehension of sequences. LSTM and BiLSTM illustrate the flexibility of RNN architectures in managing complex data relationships, highlighting its applications in DL and sequential data processing.

Where, ct as the cell state vector, output vector h(t) for the time step t

F. RNN-Based Deep Learning Approach

Applying deep neural network architectures such as LSTM and BiLSTM, this [5] aims to develop a synergistic model

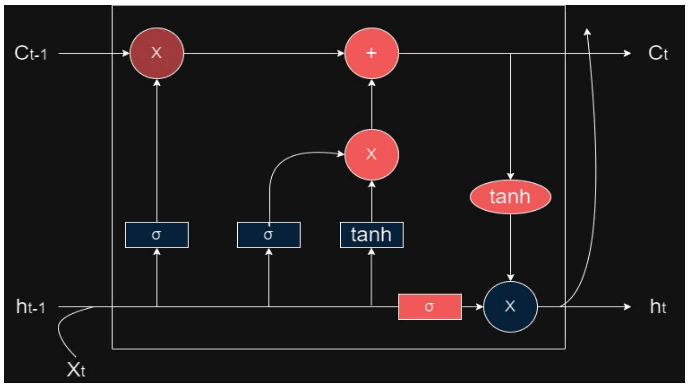


Fig. 5. The LSTM architecture
[5]

for software fault prediction. The resultant five-phase model minimizes parameters for effective layer transitions and maximizes neural network performance overall by utilizing the simultaneous functioning of BiLSTM and LSTM layers. The behavior of the DL model is determined by three important parameters: batch size, learning rate, and epochs. Finding the ideal values for these hyperparameters is an essential stage in the model-building process because they have a big influence on machine learning performance. The [5] trains the neural networks using the adaptive moments estimation (ADAM) optimization methodology, which is identical with modern deep learning optimization techniques and similar to stochastic gradient descent.

G. Performance Evaluation Metrics

Classification models are evaluated using Cohen's kappa (KE), accuracy (ACC), and area under the curve (AUC) measures. The true positive, false positive, false negative, and true negative values in a confusion matrix are the source of these measurements. When evaluating two-class classification issues, these values are important Ref. [5].

Estimated Class			
		C1(+)	C2(-)
	C1(+)	TP	FP
	∑Positive		
Actual	C2(-)	FN	TN
Class	∑Negative		

Probability of Detection , also known as recall, represents the desired outcome of correctly predicting modules containing defects, with an ideal value of one for defect prediction tasks.[5]

$$PD = \frac{TP}{TP + FN} \tag{1}$$

Probability of False Alarms, the ratio of false positives to all non-defective modules, is a performance metric ideally minimized and equal to zero in an ideal prediction task.[5]

$$PF = \frac{FP}{FP + TN} \tag{2}$$

$$ACC = \frac{TP + TN}{TP + FN + FP + TN} \tag{3}$$

$$AUC = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{4}$$

A statistic called Cohen's kappa (KE), which has a range of -1 to 1, is used to determine how well classifiers perform in comparison to chance. Lower KE values indicate a more realistic agreement. Higher KE values are closer to 1.[5]

$$KE = \frac{p0 - pc}{1 - pc} \tag{5}$$

Precision (PR): The true positive value is divided by the total of the true positive and false positive values to determine its value. The formula is used to determine the PR.[5]

$$PR = \frac{TP}{TP + FP} \tag{6}$$

F-Measure (FM): The FM is used for evaluating recall metrics and harmonic mean of accuracy, particularly if the metrics are based on incorrect information. It uses the following formula in its calculation. The formula is used to determine the FM.[5]

$$FM = 2 \cdot \left(\frac{PD \cdot PR}{PD + PR}\right) \tag{7}$$

RMSE is a commonly used measure of the distinctions between values predicted by a model and the values actually observed. The RMSE formula is shown in Equation.[42]

$$RMSE = \sqrt{\Sigma_{i=1}^{n}\left(\frac{\hat{y_i} - y_i}{n}\right)^2} \tag{8}$$

H. Selection of performance metrics

On the Receiver Operating Characteristic (ROC) curve, better outcomes are found in the top left quadrant (PD = 1, PF = 0). In software prediction, this curve shows the trade-off between the predictor's accuracy in identifying faulty (PD) and non-defective (PF) modules. ROC curves are used to evaluate the performance of the Decision Stump, Jrip, and Bayesian Logistic Regression classifiers using the AR5 dataset. The data indicates that the Decision Stump method is the most accurate at predicting figure 6. [41]

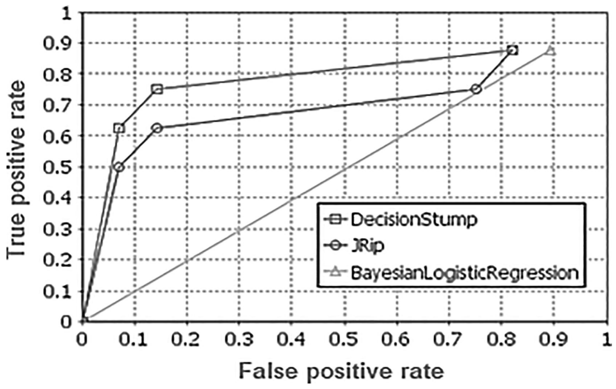


Fig. 6. Sample ROC curves for three classifiers applied to dataset AR5 [41]

Learner accuracy, an accurate measure of performance in situations with comparable outcome frequencies, is limited by differences in class distributions across datasets. Consequently, this [41] does not use ACC as a performance metric. However, software failure prediction research often uses the PD-PF pair, which offers insights into error evaluation and associated confirmation costs while finding a compromise between PF minimization and PD maximization. Through the use of the F-measure (FM), a single statistic that combines the two measures of Recall (PD) and Precision, it reduces the difficult task of separating between estimates in fundamental performance evaluation.

I. Selection of base predictors

In [42] to demonstrate how ensemble algorithms have more predictive power than base predictors, which are frequently used. Ensemble learning systems use resampling techniques such as bagging and boosting, and are classified based on their generation strategy and integration approach. Voting-based approaches that rely on simple or overwhelming majority decisions employ algebraic combinations for forecasting.

Ensemble Learning Algorithms (ELA) are combined with three-stage preprocessing techniques in the suggested system. Tukey's HSD tests, One-way ANOVA, and ranking ELAs according to AUC values are employed to evaluate significant variations in SFP model performances. Ranker search-based IG evaluation techniques are used for feature selection (FS), while ELAs and Decision Trees (DTa) are utilized for selecting the features upon which the SFP models are constructed. Additional techniques to improve ELA performance include cross-validation, INFFC approaches to address class noise, and SMOTE to address data skewness.

In Step 3, the goal is to develop a functional ensemble classifier by reducing class noise by using IG, SMOTE, and INFFC techniques. A range of performance assessment criteria and statistical approaches are applied to aggregated ELAs from Stages 1, 2, and 3 to show how performance has improved overall by fully resolving SFP problems.

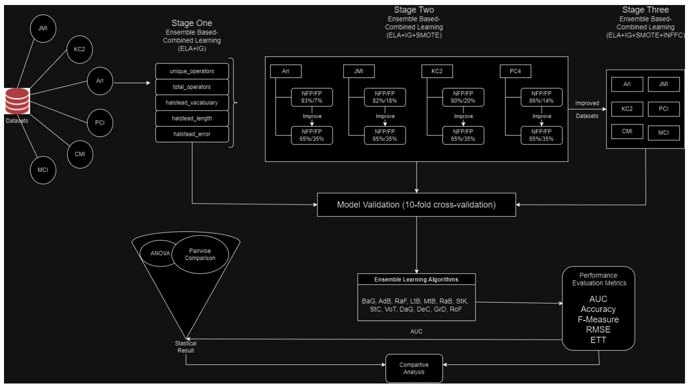


Fig. 7. A Three-Stage Based Ensemble Framework for Improving SFP [42]

J. Metrics evaluated

In [43], twenty software metrics—including well-known object-oriented measures like C and K, Henderson-Sellers, and QMOOD—were measured on several datasets. Six popular classification methods—IB1, Random Forest, J48, Bagging, Naive Bayes, and Logistic Regression—were selected based on how well they performed on average in prior research on software problem prediction. The classifier comparison made use of statistical tests, graphical techniques, and performance measurements. The study highlighted that selecting the right classifier requires considering project-specific criteria along with costs; it is not sufficient to depend only on a single variable. The review acknowledged the complex nature of project specifications in software problem forecasting and highlighted the importance of using a detailed process when selecting classifiers.

K. Neural Networks

When a problem affects more than one class, Author [36] suggests that linear regression is used to create a straight line relationship between the variables. There are now two types of linear regression investigations available: univariate and multivariate.

In situations in which the dependent variable may simultaneously store two values, producing two groups: defect-free and bug-free, logistic regression is used to predict the dependent variable's results based on one or more independent factors. In logistic regression, both univariate and multivariate analysis are useful.

This [36] combines four machine learning algorithms and the CK metric suite to measure the accuracy of defect prediction in addition to statistical methods.

Software mistakes can be predicted by an ANN using its input, hidden, and output layers. Architecture of the ANN shown in Fig. 8. This illustrates the ability of ANNs to compute and identify complex patterns.

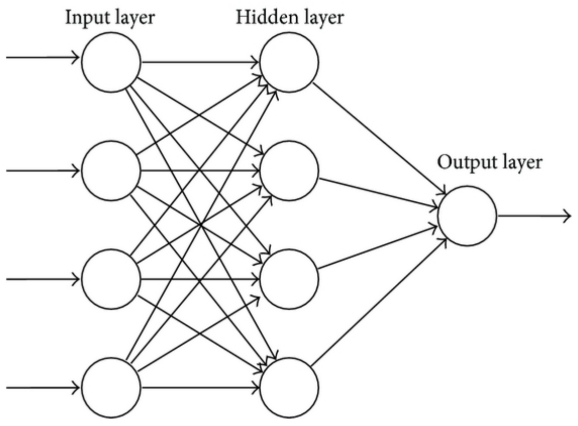


Fig. 8. Architecture of ANN
[36]

L. Standardization of the Preparation Stage

According to author [12] Standardization is applied in the first processing step of the study to extract new ranges from an equation using numerical properties, and it is also used to preprocess a heterogeneous dataset that includes MFA, CA, and other datasets. Defective cases are classified as TRUE and non-faulty cases as FALSE in order to concatenate tiny datasets into a single set. After then, variations are eliminated by cleaning.

1) Label Encoding: Label encoding, or converting TRUE and FALSE labels into numerical values, is a required preprocessing step for structured datasets in supervised learning, while it is not necessary if the dataset already has numeric label values shown in Fig. 9.

2) Applying DL Methods: Three DL algorithms are used: RBFN, BiLSTM, and LSTM. Different kinds of layers, activation functions, and hyperparameters are selected for experimentation, and iterations are carried out until desired results are obtained.

Genetic algorithms (GAs) use a population of vectors or chromosomes with binary features—1 for selection and 0 for non-selection—to find the best possible solution from a set of options. In this [44], binary strings that indicate feature selection are represented by chromosomes. Using randomly split training sets, the bagging classifier builds models, with each model using a voting method to contribute to the final classification outcome. The GA is guided in selecting features for better classification results by a fitness function that

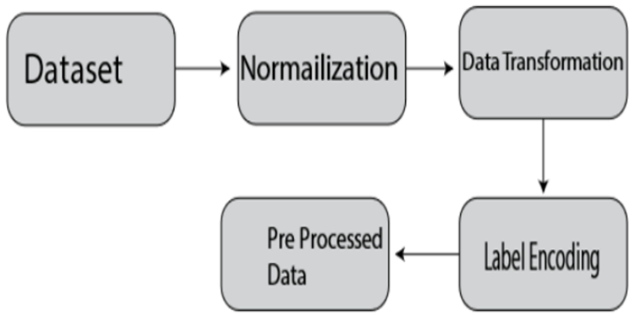


Fig. 9. Pre-Processing Steps
[12]

takes into account the number of features, feature cost, and classifier accuracy.

The analysis uses fifteen datasets from the PROMISE repository in order to understand various types of software issues. It categorizes code metrics as having problems or not, which makes machine learning easier. Well-known techniques like SVM, Naive Bayes, and others are used. They also use CNN and RNN models for images and sequences, respectively, to experiment with deep learning. Using LSTM and BiLSTM, the research suggests a unique method (RNNBDL) for improved software failure prediction. They evaluate the models according to particular standards, such as accuracy and recall. Additionally included in the analysis are ensemble techniques, which combine multiple models to improve forecasts. They use special techniques and a three-step preprocessing process to enhance results. This study looks at different classification techniques and shows how important it is to choose the right one for a certain task. They use techniques including logistic regression, linear regression, ANN, and GA. The study additionally highlights how important it is to correctly prepare data before using it for analysis.

V. Conclusion

We evaluate and compare approaches for software defect prediction using deep learning (DL) and machine learning (ML). DL is better at identifying complex patterns, even if ML is faster and more transparent—especially when using ANNs and LSTMs. The study proposes RNNBDL, an innovative technique that combines LSTM and BiLSTM to enhance predictions. When selecting algorithms developed for specific datasets, the trade-offs between interpretability and processing speed and prediction strength are highlighted. Calculate Software Errors and Bugs: Many methods, such as SVM, Naive Bayes, RF, LSTM, BiLSTM, and RBFN, are available to reduce testing costs and enhance software quality. Ensemble methods, such as stacking and SVM-PSO, provide excellent accuracy for fault prediction. Data processing and methodological decisions are essential to obtain accurate software failure rates. The study shows how the application of DL and

ML methods may enhance early error detection and reduce testing costs. As the field progresses, choosing algorithms and organizing data continue to be key elements of improving prediction models in practical software engineering situations.

References

1 I. Batool and T. A. Khan, "Software fault prediction using deep learning techniques," *Software Quality Journal*, pp. 1–40, 2023.

2 ——, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, p. 107886, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790622001744

3 G. Giray, K. E. Bennin, Ömer Köksal, Önder Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, p. 111537, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222002138

4 M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Deep learning-based defect prediction for mobile applications," *Sensors*, vol. 22, no. 13, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/13/4734

5 E. Borandag, "Software fault prediction using an rnn-based deep learning approach and ensemble machine learning techniques," *Applied Sciences*, vol. 13, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/3/1639

6 O. Alqasem and M. Akour, "Software fault prediction using deep learning algorithms," *International Journal of Open Source Software and Processes*, vol. 10, pp. 1–19, 10 2019.

7 I. Batool and T. Khan, "Software fault prediction using deep learning techniques," *Software Quality Journal*, vol. Volume 31, p. 40, 06 2023.

8 ——, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, p. 107886, 05 2022.

9 S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications*, vol. 172, p. 114595, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417421000361

10 A. Hasanpour, P. Farzi, A. Tehrani, and R. Akbari, "Software defect prediction based on deep learning models: Performance study," 2020.

11 O. Al Qasem and M. Akour, "Software fault prediction using deep learning algorithms," *International Journal of Open Source Software and Processes (IJOSSP)*, vol. 10, no. 4, pp. 1–19, 2019.

12 O. A. Qasem, M. Akour, and M. Alenezi, "The influence of deep learning algorithms factors in software fault prediction," *IEEE Access*, vol. 8, pp. 63 945–63 960, 2020.

13 S. Omri and C. Sinz, "Deep learning for software defect prediction: A survey," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 209–214. [Online]. Available: https://doi.org/10.1145/3387940.3391463

14 I. Batool and T. Khan, "Software fault prediction using data mining, machine learning and deep learning techniques: A systematic literature review," *Computers and Electrical Engineering*, vol. 100, p. 107886, 05 2022.

15 C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346–7354, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417408007215

16 I. Batool and T. Khan, "Software fault prediction using deep learning techniques," *Software Quality Journal*, vol. Volume 31, p. 40, 06 2023.

17 G. P. Bhandari and R. Gupta, "Measuring the fault predictability of software using deep learning techniques with software metrics," in *2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2018, pp. 1–6.

18 D. Sharma and P. Chandra, "Software fault prediction using machine-learning techniques," in *Smart Computing and Informatics: Proceedings of the First International Conference on SCI 2016, Volume 2*. Springer, 2018, pp. 541–549.

19 R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494614005857

20 M. Cetiner and O. K. Sahingoz, "A comparative analysis for machine learning based software defect prediction systems," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020, pp. 1–7.

21 Meiliana, S. Karim, H. L. H. S. Warnars, F. L. Gaol, E. Abdurachman, and B. Soewito, "Software metrics for fault prediction using machine learning approaches: A literature review with promise repository dataset," pp. 19–23, 2017.

22 T. Kumar and B. Booba, "A systematic study on machine learning techniques for predicting software faults," pp. 133–136, 2021.

23 R. T. Selvi and P. Patchaiammal, "Fault prediction for large scale projects using deep learning techniques," pp. 482–489, 2022.

24 C. Prabha and N. Shivakumar, "Software defect prediction using machine learning techniques," pp. 728–733, 2020.

25 C. Anjali, J. P. M. Dhas, and J. A. P. Singh, "A study on predicting software defects with machine learning algorithms," pp. 195–198, 2022.

26 A. Kumar and A. Bansal, "Software fault proneness prediction using genetic based machine learning techniques," pp. 1–5, 2019.

27 H. A. Alhija, M. Azzeh, and F. Almasalha, "Software defect prediction using support vector machine," 2022.

28 B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Software defect prediction using dynamic support vector machine," pp. 260–263, 2013.

29 M. Hammad, A. Alqaddoumi, H. Alobaidy, and K. Almseidein, "Predicting software faults based on k-nearest neighbors classification," 09 2019.

30 A. Nasser, W. Ghanem, A. Shaddad, A. Abdul-Qawy, M. Ali, A.-M. Saad, S. Ghaleb, and N. Alduais, "A robust tuned k-nearest neighbours classifier for software defect prediction," 08 2022.

31 A. Salem, K. Rekab, and J. Whittaker, "Prediction of software failures through logistic regression," *Information and Software Technology*, vol. 46, pp. 781–789, 09 2004.

32 Razauddin, S. Madhuri G, A. Oberoi, A. Vats, A. Sivasangari, and K. Siwach, "Research on efficient software defect prediction using deep learning approaches," pp. 549–554, 2022.

33 K. Wongpheng and P. Visutsak, "Software defect prediction using convolutional neural network," pp. 240–243, 2020.

34 B. Gezici and A. K. Tarhan, "Explainable ai for software defect prediction with gradient boosting classifier," pp. 1–6, 2022.

35 H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917300113

36 Y. Suresh, L. Kumar, and S. K. Rath, "Statistical and machine learning methods for software fault prediction using ck metric suite: A comparative analysis," *ISRN Software Engineering*, vol. 2014, p. 251083, Mar 2014. [Online]. Available: https://doi.org/10.1155/2014/251083

37 G. R. S. Anuradha, K. Eswara Rao, "Gradient tree boosting approach for software defect prediction," *International Journal of Advanced Science and Technology*, vol. 28, no. 20, pp. 750 –, Dec. 2019. [Online]. Available: http://sersc.org/journals/index.php/IJAST/article/view/2912

38 S. A. Ali, N. R. Roy, and D. Raj, "Software defect prediction using machine learning," *2023 10th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 639–642, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:241669703

39 R. Ponnala and D. REDDY, "Software defect prediction using machine learning algorithms: Current state of the art," *Solid State Technology*, vol. 64, pp. 6541–6556, 05 2021.

40 S. A. Ali, N. R. Roy, and D. Raj, "Software defect prediction using machine learning," pp. 639–642, 2023.

41 F. Yucalar, A. Ozcift, E. Borandag, and D. Kilinc, "Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability," *Engineering Science and Technology, an International Journal*, vol. 23, no. 4, pp. 938–950, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098619308390

42 W. Chubato, T. Li, and K. Bashir, "A three-stage based ensemble learning for improved software fault prediction: An empirical comparative study," *International Journal of Computational Intelligence Systems*, vol. 11, p. 1229, 11 2018.

43 A. Kaur and I. Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 1, pp. 2–17, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157816300222

44 R. Wahono and N. Suryana, “Genetic feature selection for software defect prediction,” *Advanced Science Letters*, vol. 20, pp. 239–244, 01 2014.