Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 47 No. 02 (2026)

# An Ensemble Feature Optimization and Xtream Learning for High-Accuracy Software Reliability Modeling

**Shaik Shakeer Basha[1], Dr.R.Satya Prasad[2], Dr.Syed Khasim[3]**

[1]Research Scholar, Department of Computer Science, Rayalaseema University, Kurnool, Andhra Pradesh, India.
[2]Professor, Department of Computer Science & Engineering, Acharya Nagarjuna University, Nagarjuna Nagar, Guntur, Andhra Pradesh, India.
[3]Professor, Department of Computer Science & Engineering, Dr.Samuel George Institute of Engineering & Technology, Markapur, Prakasam District, Andhra Pradesh, India.

***Abstract:*** Software reliability is the domain that delivers accurate, reliable, and high-quality software by estimating faults early in development. Many existing reliability approaches have failed to estimate faults, errors, and expectations in software development. The main reasons are advanced programming languages, high-level features, and complex connections among several software metrics, which have a significant impact on software development performance. These issues are addressed by presenting the Ensemble Feature Optimization and Xtream Learning (EFXLM) framework for high-accuracy software reliability modeling. The proposed EFXLM is an integrated model that combines the Whale Optimization Algorithm (WOA) to estimate failure rate, using optimized tuning parameters to achieve accurate defect detection. To improve WOA performance, a Deep Extreme Learning Machine (DELM) is proposed to reduce failure rates by leveraging multiple hidden layers. The experiments are conducted on two benchmark datasets from the NASA and PROMISE repositories to evaluate the algorithms' performance. Performance can be measured as the failure prediction rate in binary classification.
***Keywords:*** Ensemble Feature Optimization and Xtream Learning, Whale Optimization Algorithm (WOA), Deep Extreme Learning Machine (DELM), NASA, PROMISE,

## 1. Introduction

Software reliability is a critical quality attribute that reflects the ability of a software system to perform its intended functions without failure under specified conditions for a given period of time. With the increasing complexity of modern software systems, ensuring reliability has become a challenging task for developers and organizations. Traditional reliability assessment methods, which are often based on statistical models and historical failure data, may not effectively capture the dynamic behavior and nonlinear relationships present in contemporary software environments. As a result, there is a growing need for intelligent approaches that can analyze large-scale software data and provide accurate reliability predictions.

Machine learning (ML) has emerged as a powerful tool for software reliability analysis due to its ability to learn patterns from data and make data-driven predictions. ML techniques can analyze various software metrics, such as code complexity, size, change history, and fault data, to predict defects and estimate failure rates. By leveraging supervised and ensemble learning models, ML-based approaches can improve prediction accuracy, handle high-dimensional data, and adapt to evolving software systems. These capabilities enable early detection of potential failures, support proactive maintenance, and enhance decision-making in software development processes, ultimately leading to more reliable and high-quality software systems.

Failure estimation is an important aspect of software development because it enhances directly on the measurement and enhancement of software reliability, that is, the capacity of a system to function without breakdown over a particular duration. This allows developers to detect susceptible parts, comprehend fault modes, and analyze the health of the software in general by estimating both the frequency and probability of failures. The failure estimation assists in identifying any defects at an early stage in the development lifecycle, and teams can implement corrective measures before the project can be implemented. This mitigates the chances of system failures, increases user satisfaction, and the software is of quality and performance. It also helps in better planning of the testing activities as the efforts are concentrated in high risk areas thus maximizing the use

of the resources. In addition, failure estimation helps in making decisions proactively in maintenance and release management. The rate at which failures can take place, the developers and project managers can decide whether the software is ready to be deployed or needs additional testing and polishing. It can also be used to predict the future behavior of the system, to assist the organization schedule updates, utilize resources effectively and reduce downtimes. When dealing with large-scale and critical systems, proper estimation of failures is vital in ensuring the safety, reliability and continuous operation. In general, failure estimation improves the reliability of software by offering information that results in better design, fewer defects, and more reliable and resilient software systems.

**2. Literature Survey**

Kovur et al. [9] suggested a ML-based reliability assessment model of software defect prediction and model validation assessment. The presented method exploits data-driven methods in order to analyze software metrics and identify defect-prone modules more accurately and efficiently. Under the proposed framework, the software datasets are preprocessed by way of normalization, missing values, and feature selection in order to improve the quality of data and shrink dimensions. Results show that the proposed approach obtains the better results. Akram et al. [10] suggested the application of Cuckoo Search (CS) algorithm which is a nature-inspired metaheuristic optimization method to use in efficient and effective estimation of SRGM parameters. The suggested solution is based on the Levy flight-based search mechanism of Cuckoo Search which is used to explore the parameter space efficiently and prevent local optima. The algorithm approximates SRGM parameters by using candidate solutions (representing the SRGM parameters) and by updating these solutions (through fitness based evaluation) which is usually determined by error minimization criteria (like MSE). The exploration-exploitation balance of the CS algorithm also allows it to achieve the optimal parameter values more efficiently than traditional algorithms. The modeled software failure behavior and predicted reliability increase with time is then obtained based on the estimated parameters.

Behera et al. [11] discusses and evaluates the available intelligent methods to predict software reliability, with a broad spectrum of approaches, such as machine learning models, deep learning architectures, evolutionary algorithms, and hybrid optimization methods. These techniques are categorized in the survey based on their methods, data, performance measures and application domain. It also considers significant problems, such as high-dimensional data, class imbalance, model generalization and parameter optimization. Pritchard et al. [12] introduced Three-Stage Adjusted Regression Forecasting (TSARF) model of precise and powerful software defect prediction. The initial regression modeling will be used as the first step in the proposed approach to capture the initial baseline relationship between software metrics and defect occurrence. The second stage involves the application of adjustment mechanisms to optimize the predictions by fixing the residual error and solving data problems, which include class imbalance and noise. The third step incorporates an optimized forecasting model that incorporates modified outputs with extra learning strategies to increase forecast accuracy and stability. This multi-stage design would allow the model to enhance its prediction to increasingly better in the future by integrating linear and nonlinear relationships within the data.

Chennappan et al. [13] suggested an automated software failure prediction method using hybrid ML algorithms which integrates the strengths of several learning models to enhance prediction accuracy and strength. The suggested framework combines high-level preprocessing methods, such as data normalization, missing value treatment, and feature selection, to improve data quality and minimize redundancy. Hybrid learning approach is able to capture both the linear and nonlinear associations of software metrics. Intelligent optimization techniques are used to automatically select and optimize model parameters in the system, and it guarantees flexibility to various software projects. Moreover, voting or stacking can be employed to combine the predictions of more than one model to achieve better classification performance and less overfitting. Liu et al. [14] suggested a reliability estimation algorithm that involves combination of failure mechanism and ANN with the aid of Wiener processes. The suggested method integrates the data-driven learning with the stochastic modeling to enhance the accuracy and power of the reliability estimation. In this approach, a Wiener process is used to describe the stochastic time dependence of the software degradation and failure behavior, including the random fluctuations and uncertainty in system performance. The ANN is made to learn nonlinear relationships between software measures and failure modes in order to more precisely predict system reliability. The combination of the failure mechanism modeling and ANN makes the system easier to understand because the failures that are observed are attributed to the background reasons, and the Wiener process offers a probability context to time-dependent reliability modeling. Cuauhtemoc Lopez-Martin [15] looked into applying machine learning models to software

design effort and used past project data and software metrics to enhance the efficiency and reliability of the estimation. The input features that can be used in the proposed approach can be factors like the project size, complexity, staff experience, and development environment. Normalization, feature selection, and missing values are all data preprocessing methods used to optimize the performance of the models. Moreover, hyperparameter optimization and cross-validation methods are also introduced to enhance generalization and minimize the overfitting. Sedighe Ajorloo et al. [16] introduced a systematic review of machine learning in software testing, which offers a detailed analysis of the existing research tendencies and approaches. The review classifies the current research work by the nature of the ML methods used, including supervised learning, unsupervised learning, reinforcement learning, and deep learning methods. It discusses how they are used in various testing processes, such as in regression testing, performance testing, and test generation tools. Other datasets, assessment metrics, and validation strategies are also examined to determine the effectiveness of ML-based testing models in the paper. Additionally, the review notes that ML techniques have the benefits of better coverage of tests, shorter testing time, and better defect detection.

Rashid et al. [17] suggested an ANN-based software cost estimation model that is known as the CANN model, which combines the inputs generated by COCOMO with the ability of neural networks to improve the accuracy of estimations. The ANN is fed with important parameters and cost drivers in the COCOMO model, including effort multipliers and scale factors, in the proposed approach. The neural network is trained to know the nonlinear relationship between these inputs and the real software development effort. Preprocessing methods, such as normalization and feature selection, are used to enhance the performance of the models. ANN architecture has many hidden layers to ensure that it captures complex interactions between input variables, whereas training is done through optimization algorithms to reduce prediction error. Vanathi et al. [18] have introduced a hybrid optimization-based neural network model, which combines Dragonfly Optimization Algorithm (DOA) and WOA with a Multilayer Perceptron (MLP) to estimate software cost and effort better. The weights and biases of the MLP network are optimized using the Dragonfly and Whale optimization algorithms in the proposed solution to enhance the rate at which the algorithm converges and prevent local optima. To learn complicated patterns based on software project data, the hybrid DA -WOA approach strikes a balance between exploration and exploitation in the search space. The input features (project size, complexity, and cost drivers) are preprocessed using the normalization and feature selection methods to improve the efficiency of learning.

## 3. Methodology

The Ensemble Feature Optimization and Xtream Learning (EFXLM) framework is a hybrid methodology framework that aims at modeling software reliability with high accuracy using combined optimization and learning approaches. First, NASA and PROMISE repositories are analyzed by gathering software datasets and processing them through preprocessing procedures, including data cleaning, normalization, and missing values, in order to guarantee data quality. The initial step is the ensemble feature optimization strategy, which means the selection of the most relevant and informative software metrics to minimize redundancy and dimensionality. Next, Whale Optimization Algorithm (WOA) is used to approximate the failure rate of the software by optimizing the important model parameters. In WOA, the exploration and exploitation of the capabilities are used to search the best parameter values to model the failure behavior and enhance defect detection performance. At the second level, a Deep Extreme Learning Machine (DELM) is incorporated into the system to further improve the prediction accuracy and minimize the failure rates. The DELM has numerous hidden layers that model nonlinearities among features and reliability patterns in software, which is more feature-representative as compared to conventional models. WOA is optimized and the outcomes of this are fed into the DELM, which allows learning efficiency and better generalization. The model carries out both failure rate estimation and defect classification, which provides a holistic reliability analysis. The performance of the proposed approach is evaluated experimentally on NASA and PROMISE datasets to prove its efficiency, showing better accuracy, lower prediction error, and high robustness in comparison to the traditional machine learning and optimization methods.

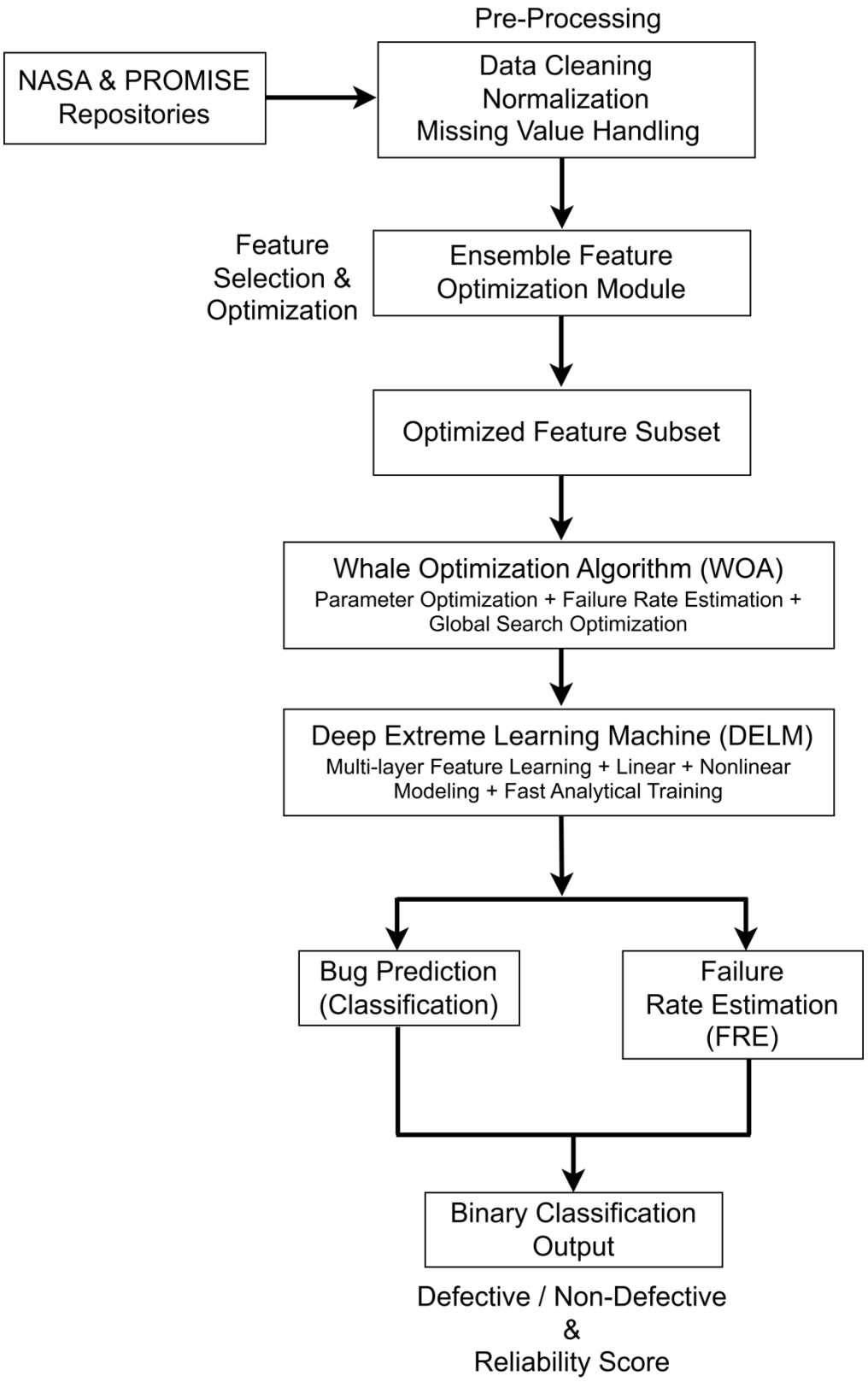


Figure 1: Architecture Diagram for Proposed Approach

***A. Whale Optimization Algorithm (WOA)***

WOA, is a nature-based metaheuristic optimization technique that is similar to the whale hunting behavior, to be more precise, the bubble-net feeding method. WOA has found its way as a global optimization technique since it is an efficient solution to complex high dimensional optimization problems with the whale being the model of social interaction and movement. The algorithm has three key processes: encircling prey, bubble-net attack

(exploration phase), and prey search (exploration phase). The exploration phase allows the whales to discover new solutions to prevent the algorithm to local optima by selecting positions randomly. Whales in the encircling process update their positions depending on the existing best solution, whereas the bubble-net strategy represents an encircling mechanism and a spiral-shaped motion toward the prey. By randomly choosing positions the exploration phase enables whales to find new solutions, preventing the algorithm to local optima. All these mechanisms make WOA a potent optimization tool because it allows balancing exploration and exploitation in a way that is effective. The mathematical equation of the proposed approach is measured as:

$$X_{t+1} = \begin{cases} X^* - A \cdot |C \cdot X^* - X_t|, & p < 0.5 \text{ and } |A| < 1 \\ X_{rand} - A \cdot |C \cdot X_{rand} - X_t|, & p < 0.5 \text{ and } |A| \geq 1 \\ |X^* - X_t| \cdot e^{bl} \cdot \cos(2\pi l) + X^*, & p \geq 0.5 \end{cases} \quad (1)$$

$X_t \rightarrow$ present position of whale; $X_{t+1} \rightarrow$ Updated position; $X^* \rightarrow$ Best solution obtained; $X_{rand} \rightarrow$ Random whale position; $A = 2a \cdot r - a \rightarrow$ Coefficient vector; $C = 2r \rightarrow$ Coefficient vector; $a \rightarrow$ Decreasing parameter (from 2 to 0); $r \rightarrow$ Random number in [0, 1]; $p \rightarrow$ Probability (0 to 1); $b \rightarrow$ Constant defining spiral shape; $l \rightarrow$ Random number in [-1, 1].

With respect to software reliability, WOA can be applied to maximize the parameters of the model and predict the breakdown of the complex software systems. Software datasets are often nonlinearly related, have numerous dimensions, and uncertainty, which are difficult to resolve using traditional optimization techniques. To solve these problems, WOA uses an effective search of the solution space to find the best parameter settings that reduce the number of predictions errors. It can be used with ML models to enhance their performance by hyper-parameter optimization, feature selection, and faster convergence. Also, the capability of WOA to move out of local minima and dynamically adapt itself to various landscapes of problems renders it appropriate in reliability models and defect prediction activities. Altogether, the Whale Optimization Algorithm is a powerful and scalable method of optimization which greatly enhances the accuracy and efficiency of the software reliability prediction systems.

***B. Deep Extreme Learning Machine (DELM)***

The DELM is a more complex version of the classic Extreme Learning Machine (ELM) which adds several hidden layers to the system to increase its learning capacity and representational power. In comparison with the more classical form of deep neural networks, in which the change of weights is calculated by the iterative use of the back-propagation algorithm, DELM is based on the simple idea of ELM: the change in weights on the input is randomized and the change in weight on the output is obtained analytically, using a method like the Moore-Penrose pseudo-inverse. DELM has a series of hierarchically layered hidden layers where each layer is a nonlinear transformation of the input data allowing the model to learn successively more abstract and meaningful features. This level architecture enables DELM to encode the patterns and dependencies of high dimension data effectively and can be of particular value when it comes to software reliability prediction where a correlation between the software metrics and the failure behavior is usually nonlinear and complex. The single unified mathematical equation is represented as:

$$\hat{Y} = H^{(L)}\beta = f\left(W^{(L)}f\left(W^{(L-1)} \ldots . f\left(W^{(1)}X + b^{(1)}\right) \ldots . + b^{(L-1)}\right) + b^{(L)}\right)\beta \quad (2)$$

$X \in R^n \rightarrow$ Input feature vector; $W^{(1)} \rightarrow$ Weight matrix of layer l; $b^{(1)} \rightarrow$ Bias of layer l; $f(\cdot) \rightarrow$ Activation function; $H^{(1)} \rightarrow$ Output of final hidden layer; $\beta \rightarrow$ Output matrix; $\hat{Y} \rightarrow$ Predicted output.

The final weighted output is represented as:

$$\beta = (H^{(L)})^{\intercal}Y \quad (3)$$

$(H^{(L)})^{\intercal} \rightarrow$ Moore-Penrose pseudo-inverse; $Y \rightarrow$ Target output.

Moreover, DELM has huge benefits of computational efficiency, scalability, and generalization over the conventional DL models. DELM is a much faster learner, because it does not rely on gradual training based on gradients, and does not compromise prediction accuracy. The multi-layer structure is also effective at the extraction of features because it incorporates both linear and nonlinear transformations, thereby increasing the capacity of the model to differentiate between defective and non-defective software modules. Also, DELM can be readily combined with the optimization algorithms like genetic algorithms or the WOA to further improve its performance by optimizing parameters and feature selection. The combination of these leads to an effective and flexible learning system that can process large, noisy, and imbalanced datasets and, in the end, increase the accuracy and reliability of software failures prediction systems.

***C. Ensemble Feature Optimization Methods***

The ensemble feature optimization techniques are those techniques that combine several feature selection and optimization techniques in order to choose the most relevant and informative features to use in a certain predictive process, like software reliability modeling. Ensemble approaches do not depend on one approach but instead integrate various strategies, which are normally filter approach, wrapper approach, and embedded approach, to take the advantage of their strengths and address their weaknesses. Filter methods (e.g., information gain, chi-square, mutual information) use the statistical properties of features to rank features, whereas wrapper methods (e.g., genetic algorithms, forward/backward selection) to rank sets of features based on model performance as a criterion. Embedded methods (e.g., LASSO, tree-based feature importance) are methods of feature selection to be used in training the model. The combination of these methods gives ensemble feature optimization a stronger and more stable selection process, with a lower probability of including irrelevant or redundant features and potentially giving the data a higher quality. The process of classification is one of the core ML tasks that is performed to categorize input data points into the existing categories based on the obtained patterns. In software reliability prediction, classification is used to classify the software modules as defective (prone to failure) and non-defective (reliable) classes. Once the optimized feature subset is obtained with the help of ensemble feature optimization, the identified features are inputted into a learning model like the XLM. The model is trained on the correlation between software measurements and defect labels and then it applies this information to detect the type of unknown data. This is done to facilitate the early detection of fault-prone modules in order to proactively maintain them and enhance the overall software quality. The classification procedure consists of mapping the output of the model to a probability score with the help of an activation function like the sigmoid function in case of binary classification. Based on a predefined threshold (typically 0.5) the result is converted to labels representing the classes of the output, larger values above the threshold representing defective modules and smaller values below the threshold representing non-defective modules.

$$F_{opt} = \arg\max_{F \subseteq X}[\alpha \cdot S_{filter}(F) + \beta \cdot S_{wrapper}(F, M) + \gamma \cdot S_{embedded}(F, M)] \quad (4)$$

$X = \{f_1, f_2, \dots, f_n\} \rightarrow$ Original feature set; $F \subseteq X \rightarrow$ Selected feature subset; $F_{opt} \rightarrow$ Optimal feature subset; $S_{filter}(F) \rightarrow$ Score from filter methods; $S_{wrapper}(F, M) \rightarrow$ Model based score; $S_{embedded}(F, M) \rightarrow$ Embedded method score; M- Machine Learning model.

$$\hat{Y} = C\left(M\left(F_{opt}\right)\right) \quad (5)$$

The fully expanded equation is represented by combining (4) and (5):

$$\hat{Y} = \sigma(M(\arg\max_{F \subseteq X}[\alpha S_{filter}(F) + \beta Accuracy\left(M(F)\right) + \gamma Importance(M, F)])) \quad (6)$$

Where $\sigma(\cdot) \rightarrow$ Classification rate ;

$$Class = \begin{cases} 1 & Defective(Failure) \\ 0 & Non-defective \end{cases} \quad (7)$$

The final step Eq: (7) represents the classification.

Besides, ensemble feature optimization reduces overfitting, generalization, and high-dimensional data effectively to optimize model performance. The datasets used in software reliability prediction, including NASA and PROMISE, usually have a large number of correlated and noisy features, which adversely affect learning algorithms. Ensemble methods solve this by combining the rankings/selections of features of several techniques, and typically they utilize a voting or weighted scheme to establish the final optimized set of features. It results in a more stable and trustworthy set of features that reflects the most important trends that contribute to software defects and failure rates. Also, ensemble optimization may be combined with metaheuristic methods such as the genetic algorithms or Whale Optimization Algorithm to further optimize the feature selection and parameter optimization. Altogether, ensemble feature optimization is an efficient and versatile method of establishing better prediction accuracy and computational efficiency in intricate software reliability modeling problems.

**4. Dataset Description**

NASA (MDP) dataset is the most popular datasets of software defect classification that consists of numerous modules of projects, such as KC1, KC2, JM1, CM1, PC1 and PC2. Together these datasets have about 20,712 software modules with 2,685 considered defective (bug-prone) and 18,027 non-defective (clean modules). All records are software components with the measures of a static code line of code, complexity, and Halstead features and binary class label (1 defective and 0 non-defective). The peculiarity of the NASA data is that the number of classes of the non-defective samples is significantly greater than the number of defective ones and therefore the data could be applied to the analysis of the strength and performance of the classification models in

predicting the software reliability.

The datasets of the PROMISE repository are also providing the greatest variety of analysis by integrating data that arrives in different software packages such as the eclipse and apache besides the NASA derived datasets. The combined data in some common PROMISE subsets (e.g. KC1, JM1, Eclipse versions and Apache projects) is usually some 14,000 to 15,000 records, comprising of approximately 2,600-2,700 defective records and the rest 11,000+ non-defective records. These data sets are composed of process and non-process data, and can make use of a larger and more diverse feature space to do classification. Just like NASA datasets, PROMISE datasets have also the problem of class imbalance and high dimensionality, which require application of feature selection and optimization techniques. In general, both data sets represent a rich basis of training and testing machine learning models to predict software defects and integrated software reliability analysis.

**5. Performance Metrics**

To evaluate the utility of software reliability prediction models, the performance measures are required to measure the degree of accuracy in the detection of defective and non-defective modules and the prediction of the failure behavior. Accuracy is used to measure the general accuracy of the model which means the percentage of model predictions that are accurate. Precision, a measure of the model correctly identifying defective modules without the false alarm, and recall (sensitivity), a measure of the model identifying all the actual defected modules. There is a balanced evaluation of F1-score in that it adds the precision and recall and is, consequently, particularly convenient with imbalanced data, such as NASA and PROMISE. There is also the error rate that indicates the percentage of inaccurate prediction, which indicates weak areas in the model. In the case of estimating failure rates, regression measures, which include Mean Absolute Error (MAE), root mean square error (RMSE), and R2 score are applied to determine the accuracy of prediction, deviation, and goodness of fit. A combination of these is a means of obtaining a general evaluation of the classification and regression performance that will render the proposed model reliable and robust.

**6. Results and Discussions**

Table 1 shows the relative performance of PSO, OA-XLM, and the proposed EFXLM model on the NASA data in terms of failure rate estimation with the evaluation metrics of MAE, RMSE and R 2 score. Particle Swamp Optimization (PSO) model has an MAE of 0.128 with an RMSE of 0.169 and has a moderate level of prediction with a visible variation to actual failure values. The high value of R 2 at 0.91 indicates that the model can account 91 percent of variance in the data, which indicates a good predictive power rather than an optimal one. The OA-XLM model is also better than the PSO with a reduced MAE 0.079 and RMSE 0.138 as it shows an increased accuracy and reduced error in prediction. Also, the R 2 score goes up to 0.96, which means that the predicted and actual values are more strongly correlated, and the model fits them better. The suggested EFXLM model performs much better than both PSO and OA-XLM in all measures with the smallest MAE (0.061) and RMSE (0.113), which signifies low error rate in prediction and greater accuracy in estimating the failure rate. The maximum R 2 value of 0.98 indicates that the model is highly predictive, with an excellent predictive performance and a high generalization power. This enhanced performance can be based on the incorporation of ensemble feature optimization, Whale Optimization Algorithm to tune the parameters, and Deep Extreme Learning Machine to learn deep features. The combination of these elements improves the capability of the model to replicate intricate patterns, minimize redundancy, and maximize learning leading to more precise and dependable software failure rate forecasting.

**Table 1: Failure Rate Estimation Performance for NASA Dataset**

| Algorithms | MAE | RMSE | $R^2$ Score |
|---|---|---|---|
| PSO | 0.128 | 0.169 | 0.91 |
| OA-XLM | 0.079 | 0.138 | 0.96 |
| Proposed EFXLM | 0.061 | 0.113 | 0.98 |

Table 2 shows the estimation failure rate of PSO, OA-XLM, and the developed EFXLM model based on MAE, RMSE, and R 2 score on the PROMISE dataset. The approach based on the PSO shows an MAE of 0.136 and RMSE of 0.161, which is rather high in terms of the prediction error and deviation between the actual values of failure, and the R2 value of 0.91 indicates that the approach can explain 91 percent of the variance in the data.

This represents a sensible amount of predictive ability as well as illustrates the constraints of managing the intricacy and variability of PROMISE datasets. OA-XLM model is much better in performance since the MAE of 0.083 and RMSE of 0.121 are less accurate and the model shows less error of estimation. The R 2 value of 0.97 demonstrates that the model is strongly correlated with predicted and actual values and thus it is able to reflect nonlinear features in the data using optimized feature learning. The EFXLM model proposed further improves the performance and has the least results compared to all the other methods with the lowest MAE (0.071), and RMSE (0.101) meaning that there is very little error in prediction and that the estimate of the failure rate is high. R 2 = 0.99 indicates that the model has a near perfect fit, that is, it accounts 99 percent of the variation in the data and has an excellent generalization ability. This high performance is mainly attributed to the combination of ensemble feature optimization, Whale Optimization Algorithm to carry out the effective parameter tuning and Deep Extreme Learning Machine to pursue deep hierarchical learning. These elements interact to minimize redundancy of features, to fine-tune or optimize model parameters and to find intricate patterns in software data. Consequently, EFXLM framework offers a very precise, smooth, and solid answer to software reliability forecasting, especially in varied and high-dimensional information such as PROMISE.

**Table 2: Failure Rate Estimation Performance for PROMISE Dataset**

| Algorithms | MAE | RMSE | $R^2$ Score |
|---|---|---|---|
| PSO | 0.136 | 0.161 | 0.91 |
| OA-XLM | 0.083 | 0.121 | 0.97 |
| Proposed EFXLM | 0.071 | 0.101 | 0.99 |

Table 3 shows the results of the classification of PSO, OA-XLM, and proposed EFXLM model on the NASA dataset based on several important evaluation metrics including precision, recall, specificity, accuracy and F1-score. The PSO-based model has moderate performance with precision (0.89), recall (0.91), and accuracy (0.89) that do not illustrate that the model is able to determine defective modules reasonably, but still gives a significant false positive and false negative rate. Its specificity of 0.90 indicates a reasonable capacity to accurately classify non-defective modules, although overall performance is poor because it is unable to completely determine intricate associations in software data that is high-dimensional. These results are much better with the OA-XLM model which gives precision (0.96), recall (0.95) and accuracy (0.97) and has a better performance in detecting defective modules and in minimizing misclassification. The increased specificity (0.96) and F1-score (0.98) point to the high level of sensitivity and precision balance, which points to the efficiency of Xtream Learning in the process of nonlinear pattern modeling. The proposed EFXLM model is even more effective in the classification performance and best results are obtained in all metrics, with precision (0.98), recall (0.98), specificity (0.99), accuracy (0.99), and F1-score (0.99). These values show that the model is capable of detecting defective and non-defective modules with a very small margin of error and leading to almost perfect classification. The high precision means that there are very few false positives, whereas the high recall means that nearly all the defective modules are identified. The enhanced specificity demonstrates a high level of capability to identify non-defective modules, and the F1-score proves the balanced, strong model. This high performance is explained by the combination of feature optimization of ensembles, parameter tuning of WOA, and deep learning using DELM, which are capable of optimizing feature selection, learning parameters, and complex data patterns. Consequently, the EFXLM framework offers a very dependable and precise answer to software defects categorization in the NASA database.

**Table 3: Classification Performance Algorithms on NASA Dataset**

| Algorithms | PSO | OA-XLM | Proposed EFXLM |
|---|---|---|---|
| Precision | 0.89 | 0.96 | 0.98 |
| Recall | 0. 91 | 0.95 | 0.98 |
| Specificity | 0.90 | 0.96 | 0.99 |
| Accuracy | 0.89 | 0.97 | 0.99 |
| F1 Score | 0.91 | 0.98 | 0.99 |

Table 4 shows the performance of PSO, OA-XLM, and the proposed EFXLM model in the classification of the PROMISE data set based on precision, recall, specificity, accuracy, and F1-score. The PSO-based model has a reasonable detection of defective module with a baseline performance of (0.92), recall of (0.91), and accuracy of (0.92). It has a specificity of 0.90 indicating that not all non-defective modules are correctly identified as defective whereas the F1-score of 0.92 indicates an intermediate level of precision and recall. OA-XLM model exhibits some improvement with precision (0.97), recall (0.96) and accuracy (0.98) indicating some improvement in detecting defective and non-defective modules. The larger specificity (0.97) suggests that clean modules are better discriminated, and the F1-score (0.98) shows the good balance between detection accuracy and consistency, which means that Xtream Learning is effective in harvesting complex data patterns. The EFXLM model is also faster than all other approaches, as it has the highest values in all metrics with precision (0.98), recall (0.99), specificity (0.99), accuracy (0.99), and F1-score (0.99). The results suggest almost perfect classification performance with the model being able to detect almost all defective modules (high recall) and has very low false positives (high precision). Its great specificity shows a high capability to correctly categorize non-defective modules, and the high F1-score proves a well-balanced and strong model. This better performance is explained by the synergistic effect of ensemble feature optimization, Whale Optimization Algorithm to tune the best parameters and Deep Extreme Learning Machine to learn the deep features. Combined, these elements can help the model to manage the complexity, high dimensionality and variability of the PROMISE data with accuracy and reliability in classifying software defects.

**Table 4: Classification Performance Algorithms on PROMISE Dataset**

| Algorithms | PSO | OA-XLM | Proposed EFXLM |
|---|---|---|---|
| Precision | 0.92 | 0.97 | 0.98 |
| Recall | 0. 91 | 0.96 | 0.99 |
| Specificity | 0.90 | 0.97 | 0.99 |
| Accuracy | 0.92 | 0.98 | 0.99 |
| F1 Score | 0.92 | 0.98 | 0.99 |

## Conclusion

This research suggested a high-precision software reliability modeling framework based on a combination of optimization and deep learning and created a single system. The framework is good at integrating ensemble feature optimization to choose the most useful software metrics, the WOA in order to estimate failure rates and tune parameters accurately, and the DELM in order to learn complex nonlinear relationships using multi-layer learning. Through these elements, the proposed model improves the ability to detect defects and minimizes prediction errors, which deal with high-dimensional and unbalanced software datasets. The experimental assessment of the benchmark datasets in NASA and PROMISE repositories proves that the proposed EFXLM framework is much more effective than the current methods in terms of failure rate estimation and binary classification with respect to its performance. The model is robust and effective in the real world because of its high prediction accuracy, reduced error rates and better generalization. The combination of WOA and DELM makes it possible to optimize the process efficiently and learn many features, which will lead to the accurate detection of harmful and healthy software modules. Altogether, EFXLM framework offers a scalable, dependable, and smart software reliability prediction solution, which allows to predict early faults, make better decisions, and improve software quality.

## References

[1] P. B. Baronia and C. Gupta, "Software Defect Prediction for Reliability Analysis Using Machine Learning Approach," 2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG), Indore, India, 2023, pp. 1-5, doi: 10.1109/ICTBIG59752.2023.10456297.

[2] A. Vashishtha and S. K. Sharma, "Estimation of Software Reliability Testing Using Machine Learning Techniques," 2025 International Conference on Intelligent Control, Computing and Communications (IC3), Mathura, India, 2025, pp. 502-506, doi: 10.1109/IC363308.2025.10956618.

[3] S. S. Sai Priyanka, B. P. Goud, M. Pandita, K. P. Kumar, P. Sahith and J. Shiva, "A Data Driven Approach to Software Quality Estimation Using Modern Machine Learning Techniques," 2025 5th International

Conference on Intelligent Technologies (CONIT), HUBBALI, India, 2025, pp. 1-7, doi: 10.1109/CONIT65521.2025.11167139.

[4] A. Ouertani, O. Krini and H. J. Börcsök, "A Practical Approach for Reliability Prediction of Safety Critical Software Using Multi-Model Ensemble Techniques," 2023 7th International Conference on System Reliability and Safety (ICSRS), Bologna, Italy, 2023, pp. 498-506, doi: 10.1109/ICSRS59833.2023.10381372.

[5] A. Kaur, P. S. Sandhu and A. S. Bra, "Early Software Fault Prediction Using Real Time Defect Data," 2009 Second International Conference on Machine Vision, Dubai, United Arab Emirates, 2009, pp. 242-245, doi: 10.1109/ICMV.2009.54.

[6] Lee, D.H.; Chang, I.H.; Pham, H. Software Reliability Growth Model with Dependent Failures and Uncertain Operating Environments. Appl. Sci. 2022, 12, 12383. https://doi.org/10.3390/app122312383

[7] Saxena, P.; Ram, M. Two phase software reliability growth model in the presence of imperfect debugging and error generation under fuzzy paradigm. Math. Eng. Sci. Aerosp. (MESA) 2022, 13, 777–790.

[8] H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth models with normal failure time distributions," Reliability Engineering & System Safety, vol. 116, pp. 135–141, Aug. 2013, doi: https://doi.org/10.1016/j.ress.2012.02.002.

[9] K. M. Kovur, H.-U.-R. Shaik, A. K. Verma, and A. Srividya, "Machine Learning-Based Reliability Evaluation for Software Defect Prediction and Model Validation Assessment," International Journal of Reliability Quality and Safety Engineering, vol. 32, no. 03, Nov. 2024, doi: https://doi.org/10.1142/s0218539324500578.

[10] K. M. Kovur, H.-U.-R. Shaik, A. K. Verma, and A. Srividya, "Machine Learning-Based Reliability Evaluation for Software Defect Prediction and Model Validation Assessment," International Journal of Reliability Quality and Safety Engineering, vol. 32, no. 03, Nov. 2024, doi: https://doi.org/10.1142/s0218539324500578.

[11] Behera, A.K., Chaudhury, P. & Dash, C.S.K. A comprehensive survey on intelligent software reliability prediction. Discov Computing 28, 90 (2025). https://doi.org/10.1007/s10791-025-09597-z

[12] S. Pritchard, B. Mitra and V. Nagaraju, "Three-Stage Adjusted Regression Forecasting for Software Defect Prediction," 2024 Annual Reliability and Maintainability Symposium (RAMS), Albuquerque, NM, USA, 2024, pp. 1-6, doi: 10.1109/RAMS51492.2024.10457812.

[13] R. Chennappan and Vidyaathulasiraman, "An automated software failure prediction technique using hybrid machine learning algorithms," Journal of Engineering Research, vol. 11, no. 1, p. 100002, Mar. 2023, doi: https://doi.org/10.1016/j.jer.2023.100002.

[14] D. Liu et al., "A reliability estimation method based on combination of failure mechanism and ANN supported wiener processes," Heliyon, vol. 10, no. 4, p. e26230, Feb. 2024, doi: https://doi.org/10.1016/j.heliyon.2024.e26230.

[15] Cuauhtemoc Lopez-Martin, "Machine learning models for predicting software design effort," Science of Computer Programming, vol. 248, pp. 103385–103385, Aug. 2025, doi: https://doi.org/10.1016/j.scico.2025.103385.

[16] Sedighe Ajorloo, Amirhossein Jamarani, Mehdi Kashfi, Mostafa Haghi Kashani, and Abbas Najafizadeh, "A systematic review of machine learning methods in software testing," Applied soft computing, pp. 111805–111805, May 2024, doi: https://doi.org/10.1016/j.asoc.2024.111805.

[17] C. H. Rashid, I. Shafi, B. H. A. Khattak, M. Safran, S. Alfarhood, and I. Ashraf, "ANN-based software cost estimation with input from COCOMO: CANN model," Alexandria Engineering Journal, vol. 113, pp. 681–694, Dec. 2024, doi: https://doi.org/10.1016/j.aej.2024.11.042.

[18] D. Vanathi, K. Anusha, A. Ahilan, and A. Salinda Eveline Suniram, "Software cost and effort estimation using dragonfly whale optimized multilayer perceptron neural network," Alexandria Engineering Journal, vol. 103, pp. 30–37, Sep. 2024, doi: https://doi.org/10.1016/j.aej.2024.04.043.