International Journal of Computer Applications (0975 – 8887)
Volume 186 – No.59, December 2024

Hybrid ANN and Fireworks Algorithm for Real-Time Software Reliability Prediction

Ivy Botchway
University of Mines and Technology
Tarkwa, Ghana

Felix Larbi Aryeh
University of Mines and Technology
Tarkwa, Ghana

Boniface Kayode Alese
Federal University of Technology
Akure, FUTA, Nigeria

ABSTRACT
Reliability is a critical factor for assessing software quality, as it measures the software's ability to perform its intended functions without failure. In recent years, research has focused on developing more robust models for predicting software reliability. This study explores a hybrid approach for software reliability prediction by integrating Artificial Neural Networks (ANN) with the Fireworks Algorithm (FWA) and ensemble learning techniques. By leveraging FWA's optimization capabilities, the ANN's weights and biases are fine-tuned, and predictions are further refined using ensemble models consisting of Random Forest and Decision Tree algorithms. Using a real-time dataset of execution time and detected faults, the hybrid model was trained and evaluated, achieving high prediction accuracy, with the ensemble model yielding an R^2 of 0.972 and FWA optimization achieving an MSE of 0.0369 after 50 generations. The results demonstrate that combining ANN, FWA, and ensemble learning can significantly improve prediction accuracy and model reliability. Future work aims to expand this approach by incorporating additional models, exploring dynamic FWA tuning, and adapting the method across various software environments.

General Terms
Fireworks Algorithm, Artificial Neural Network, Ensemble Learning Technique, Random Forest, Decision Tree.

Keywords
Software reliability, Prediction model, Gaussian Mutation, Real-Time Systems, Machine learning model.

1. INTRODUCTION
The increasing reliance on software for various human activities has led to rapid growth in software development. Governments and businesses adopt software due to its efficiency and effectiveness, helping boost revenue. However, researchers note that increased software usage correlates with a higher likelihood of failures [2]. Given software's pivotal role in providing a competitive edge, the need for reliable software becomes critical in building trust between organizations and their customers [4]. Reliability, as defined by the American National Standards Institute (ANSI), is the ability of software to function effectively within a specific environment and timeframe without encountering breakdowns [19]. It includes attributes such as freedom from failure, long lifespan, and ease of repair, making it a crucial aspect of software quality [15][5].

Recently, Artificial Neural Networks (ANN) have gained prominence in predicting software reliability. ANNs are machine learning models inspired by the neural structure of the human brain, consisting of interconnected artificial neurons. These networks model complex relationships between inputs and outputs and are trained using historical data. The learning process continues until an optimal solution is achieved, making ANNs particularly effective for prediction tasks.

To further enhance the prediction performance, this study incorporates Ensemble Learning, a technique that combines multiple predictive models such as Decision Tree, K-Nearest Neighbor (KNN), Recurrent Neural Network (RNN), Random Forest, Support Vector Machines (SVM), and Convolutional Neural Network (CNN) to achieve better accuracy and robustness [17]. This study uses Random Forest and Decision Tree, both effective for dataset classification. Random Forest improves accuracy by generating child nodes randomly and selecting the most frequent classification results [14]. Decision Tree, a popular supervised learning method, is an iterative top-down approach with root, decision, and leaf nodes, where the root node represents the dataset's most predictive attribute [6]. By utilizing Ensemble Learning, the proposed approach reduces variance, mitigates overfitting, and strengthens the reliability predictions.

In addition, the Fireworks Algorithm (FWA), a recent optimization technique based on swarm intelligence, plays a critical role in optimizing the ANN models. The research focuses on real-time systems as such systems highlight the critical need for software reliability in time-sensitive applications such as aerospace, medical devices, or autonomous vehicles. Predicting reliability for these systems involves dynamic and complex failure modes, providing an interesting challenge for optimization algorithms like FWA.

FWA simulates the behavior of fireworks exploding in the night sky, where the sky represents the solution space, and the explosion represents the search for optimal solutions [3]. FWA generates and evaluates random fireworks according to an objective function [20]. Fireworks with better fitness values create more sparks with smaller explosion amplitudes, improving the chances of finding the best solution. The Gaussian mutation operation introduces diversity to the population after each explosion, and the next generation of fireworks is selected based on performance. This iterative process continues until the optimum solution is found.

This research proposes a novel methodology combining ANN, FWA, and Ensemble Learning for software reliability prediction. By leveraging the strengths of FWA in optimizing ANN weights and biases, and enhancing performance through ensemble techniques, the model aims to improve prediction accuracy. The paper is structured as follows: Section II reviews related work. Section III details the methodology for reliability prediction, Section IV shows the implementation, Section V discusses the experimental results, and Section VI concludes the study and outlines future research directions.

2. RELATED WORKS

Over the years, numerous software reliability growth models have been proposed by researchers to improve the development of reliable and high-quality software. In 1991, a study [11] utilized a feedforward neural network to predict software reliability growth. The researchers used real datasets from three different software projects to compare the model's performance with existing reliability models. The findings indicated that neural networks consistently performed well in prediction tasks, with further validation through statistical methods. The authors also suggested exploring the use of recurrent neural networks in future research.

In 2006, another study [18] focused on software reliability prediction using neural networks. The researchers trained their model on real-world software failure data from the John Musa Bell laboratory, employing a backpropagation algorithm to develop a weighted combinational model. The analysis demonstrated that the model could learn from software failure history and adapt its weights when encountering new failures.

A 2012 study [3] proposed a neural network-based model for software reliability prediction using a feedforward approach. The input parameters were derived from software execution time and encoded using exponential and logarithmic functions. The model was tested on eighteen software failure datasets, and the results demonstrated its efficiency and predictive accuracy. In the same year, researchers [2] introduced a hybrid model combining a neural network with a simulated annealing algorithm for reliability prediction. Their results showed that this hybrid approach was highly accurate in predicting software failures.

In 2019, researchers [1] once again utilized a feedforward neural network model to predict software reliability growth, this time using datasets from various real-world software projects. The model successfully predicted the future cumulative faults of the software.

Research has also been done on the use of ensemble learning for software reliability prediction. In 2009, research [18] was conducted to develop a non-parametric software reliability prediction system using neural network ensembles to address the limitations of traditional parametric models, such as nonhomogeneous Poisson process (NHPP) models. The researchers explored how system architecture affects performance and compared the ensemble approach to single neural networks and NHPP models. The results indicated that combining multiple neural networks significantly improves predictability, offering a promising alternative to conventional methods.

A 2024 study also [8] investigates software reliability prediction through ensemble learning enhanced by random hyperparameter optimization. It introduces a regression model that utilizes various base learners, such as Ridge and Support Vector Regressor, with Ridge serving as the combiner. The study evaluates the model's performance against benchmark datasets, showcasing its superior accuracy over existing models by effectively tuning hyperparameters and minimizing bias and variance. This approach aims to advance the predictability of software reliability compared to traditional methods.

Another study [9] in 2024 investigates software reliability prediction using ensemble learning techniques such as bagging, boosting, and stacking. The model combines multiple machine learning algorithms to predict software failures based on the Mean Time Between Failures (MTBF) using Musa's dataset and classifies software defects using NASA's dataset. The study showcases the model's superior performance, achieving 94% prediction accuracy and 97% classification accuracy, highlighting the effectiveness of ensemble methods in improving the reliability of software prediction compared to traditional approaches.

These studies collectively highlight that neural networks have become a promising alternative for software reliability prediction and modeling. Building on this, the current research adopts the Fireworks Algorithm (FWA), an optimization technique introduced in 2010 and applied across various fields for optimizing the weights and biases of an Artificial Neural Network (ANN), leveraging ensemble learning techniques to further enhance software reliability prediction. This combination significantly boosts prediction accuracy.

3. RESEARCH METHODOLOGY

3.1 Fireworks Algorithm

The Fireworks Algorithm (FWA) starts by generating initial samples and selects the next generation of explosions. The initial explosion occurs at a specified location, n, and sparks are generated and evaluated based on their final positions. The algorithm iterates until the optimal solution is found. If the optimal location is not identified, new positions are selected from the current sparks to generate the next set of explosions. A flowchart of the operation of FWA is shown in Figure 1.

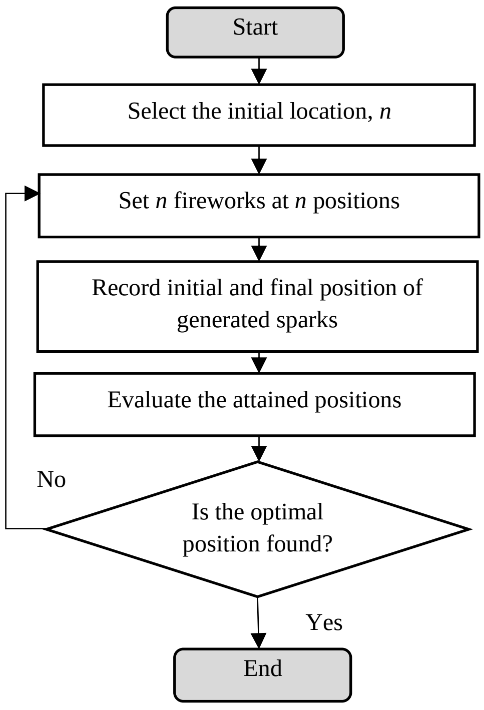
Start
Select the initial location, n
Set n fireworks at n positions
Record initial and final position of generated sparks
Evaluate the attained positions
No
Is the optimal position found?
Yes
End

Fig. 1: Operation of Firework Algorithm

Each firework explosion produces a unique shape based on its design. Hence, they possess unique explosive characteristics depending on the set-off location. These characteristics include a narrow explosion with few sparks and a wide explosion with more sparks. According to [18], fireworks exhibit two dominant behaviors: a well-designed firework produces a wide explosion with densely centered sparks, while a poorly designed one generates a narrow explosion with scattered sparks as shown in Figure 2.

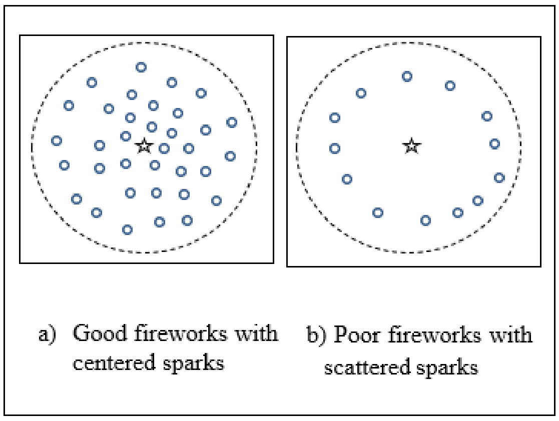
a) Good fireworks with centered sparks
b) Poor fireworks with scattered sparks

Fig. 2: Behaviors of Fireworks

In the context of search algorithms, a "good" firework, as illustrated in Figure 2a, suggests proximity to an optimal solution due to the large number of sparks that can be used in the search process. On the other hand, a "poor" firework, as shown in Figure 2b, indicates that the search is further from the optimal solution, with fewer and more dispersed sparks to guide the search.

The research employed the following Fireworks Algorithm (FWA) strategies to optimize the weights and biases.

3.1.1 Explosion Sparks Strategy

This strategy forms the core of the algorithm, generating sparks around an exploding firework. It consists of three main components: explosive strength, explosive amplitude, and explosive displacement.

Explosive Strength: This technique assigns a number of sparks to the firework, which is crucial for evaluating its amplitude and optimal value. The sparks generated by a firework, xi, are given by Equation (1):

$$s_i = m * \left(\frac{y_{\max} - f(x_i) + \varepsilon}{\sum_{i=1}^{n} (y_{\max} - f(x_i)) + \varepsilon} \right) \quad (1)$$

Where:

s_i = sparks generated by firework,

m = a constant for the total number of generated sparks,

$y_{\max}$ = worst fitness value in the current generation,

$f(x_i)$ = fitness value for an individual firework, x_i.

ε = small value to prevent division by zero.

Explosive Amplitude: In optimization, the best fitness values are typically found near the optimal solution. The explosive amplitude adjusts the explosion's width based on the current fitness value, converging towards the optimal solution. It is calculated using Equation (2):

$$A_i = \hat{A} * \left(\frac{f(x_i) - y_{\min} + \varepsilon}{\sum_{i=1}^{n} (f(x_i) - y_{\min}) + \varepsilon} \right) \quad (2)$$

Where:

A_i = amplitude of each firework,

Â = constant for maximum explosion amplitude,

ymin = best fitness value,

f(xi) = fitness value for an individual xi,

ε = value to prevent zero-division errors.

Explosive Displacement: This phase generates random values within the explosive amplitude range, ensuring diverse exploration of the solution space.

3.1.2 Gaussian Mutation Sparks Strategy

The Gaussian mutation strategy increases population diversity by generating additional sparks using the Gaussian distribution. This introduces variation into the population, helping to avoid local optima.

3.1.3 Mapping Strategy

The mapping strategy ensures that sparks generated near the boundaries of the solution space do not do not build up outside it. If a spark is outside the solution space, it is mapped back within the boundaries using modular arithmetic. The boundary constraints are defined by Equation (3):

$$x_i = x_{\min} + |x_i| \% (x_{\max} - x_{\min}) \quad (3)$$

Where:

x_i = position of sparks outside the solution space,

xmin = minimum boundary of the spark's position,

xmax = maximum boundary of the spark's position,

% = modular arithmetic operation.

3.1.4 Selection

Sparks for the next generation are selected from the explosive spark strategy and the Gaussian spark strategy using a distance-based selection method. The best spark is chosen first, followed by the selection of remaining sparks based on their Euclidean distance from each other, as calculated by Equation (4):

$$R(x_i) = \sum_{j=1}^{K} d(x_i, x_j) \quad (4)$$

Where:

xi and xj = distance between one spark from the other,

K = set of all sparks generated by both the explosive and Gaussian strategies.

This selection process ensures a diverse set of sparks for the next iteration, optimizing the search for the global solution.

3.2 Ensemble Learning

The ensemble learning paradigm, which integrates multiple base learners, is a robust approach for enhancing prediction accuracy. This section offers an overview of three widely utilized ensemble methods applied in this research.

3.2.1 Artificial Neural Networks

The structure of an Artificial Neural Network (ANN) is multi-layered, with the middle layer composed of several simple non-linear functions that mimic the behavior of biological neurons. ANNs can be visualized as a weighted directed graph, where the artificial neurons serve as nodes, and the directed edges between them represent the weights. Each neuron in an ANN, as depicted in Figure 3, receives a set of inputs, x_1, x_2, ..., xn, multiplies them by their corresponding weights, and produces an output, O, as shown in Equation (5):

$$O = f\left(\sum_{i=1}^{n} w_i \cdot x_i\right) \quad (5)$$

Where:

w_i= weight,

i and n = integers,

f = activation function.

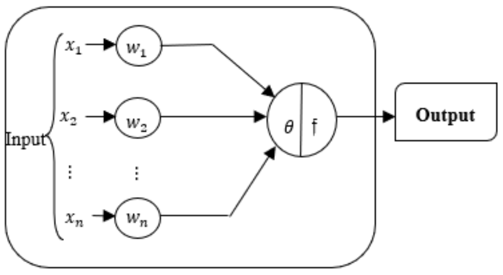

Fig. 3: Artificial Neuron

The weights in the network determine the strength of the connections between neurons, and the activation function is responsible for producing the final output. The neurons are arranged in layers, with the perceptron being a single-layered neuron. The number of neurons in the input layer corresponds directly to the number of input attributes, while the number of neurons in the output layer corresponds to the desired output values.

The weights between neurons in ANN are calculated using Equation (6):

$$w_i = (x_i * h_i) + h_i * O_i) \qquad (6)$$

Where:

wi= weight,

xi = input,

hi = hidden layer output,

Oi = final output.

This structure allows the network to process complex patterns and make predictions by adjusting weights through training.

3.2.2 Decision Tree

Decision trees are widely used for decision-making and problem-solving, with criteria acting as interconnected nodes to form a tree-like structure [10]. It is organized hierarchically, comprising a parent node and two child nodes [12]. Each branch represents an attribute that must be satisfied to progress to the next branch, ultimately leading to the leaf node. Each parent node includes a split condition that identifies a predictor and a cutoff value, aiming to minimize impurities in the child nodes. The reduction in impurities resulting from this split is known as information gain, which is evaluated using various criteria. The tree-building process continues until a leaf node is reached, which signifies the predicted class of the response variable [7].

3.2.3 Random Forest

Random Forest (RF) is an algorithm that employs a recursive binary split method to navigate through a tree structure for classification and regression tasks [16]. This algorithm offers several benefits, such as achieving relatively low error rates, excellent classification performance, and the ability to efficiently handle large training datasets while effectively estimating missing data. The RF method enhances accuracy by randomly generating child nodes for each node. It constructs a decision tree with root nodes, internal nodes, and leaf nodes, selecting attributes and data randomly according to established guidelines. The root node sits at the top of the tree, while internal nodes are branching points with at least two outputs and one input. The leaf node, or terminal node, is the final node with only one input and no outputs. The decision tree process starts by calculating the entropy value to assess the impurity level of the attributes and determine information gain [13].

4. EXPERIMENTAL DATA SETUP

4.1 Dataset

Analyzing software reliability growth models requires multiple datasets, but collecting them can be difficult due to the confidentiality surrounding software failure dataset. In this research, we utilized a real-time dataset from a software project represented as D in Equation (7), which reflects the accumulated execution time and the corresponding number of faults.

$$D = \{(N_i + t_i),\ i = 1, 2 \ldots n\} \qquad (7)$$

Where:

Ni = accumulated number of software failures,

ti = the execution time.

The dataset encapsulates the failure history of the software and was tested over thirty weeks, with a cumulative debugging time of 65.77 hours and 472 failures. The network was trained using execution time as input and fault count as output. After training, the network predicted the total number of faults based on the final execution time during testing. This dataset is detailed in Table 1.

Table 1. Dataset Used for Experiment

Weeks	Execution Time of CPU (hours)	Number of detected Faults
1	4.67	35
2	6.02	76
3	9.56	89
4	11.25	106
5	13.79	117
6	16.23	125
7	18.67	179
8	21.08	192
9	22.26	203
10	25.40	248
11	27.49	283
12	29.43	291
13	31.51	325
14	34.41	337
15	37.96	343
16	39.34	349
17	41.70	358
18	41.96	372
19	44.89	379
20	45.76	385
21	46.72	397
22	48.99	401
23	54.78	413
24	56.93	419

25	57.84	420
26	58.57	429
27	60.05	442
28	60.91	449
29	63.78	458
30	65.77	472

4.2 Normalization, Pre-processing and Splitting of Dataset

The data used for prediction was first cleaned and organized into separate datasets, with 70% allocated for training and 30% for testing. The dataset was then normalized using the min-max normalization technique, as shown in Equation (8), to scale it between 0 and 1:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad (8)$$

Where:

X_{norm} = Normalized value,

X = Original data value,

X_{min} = Minimum value,

X_{max} = Maximum value.

The model's input consists of the software's execution time, while the output is the predicted number of failures. The normalized dataset is detailed in Table 2.

Table 2. Normalized Dataset

Weeks	Normalized Execution Time	Normalized Detected Faults
1	0.000000	0.000000
2	0.022095	0.093822
3	0.080033	0.123570
4	0.107692	0.162471
5	0.149264	0.187643
6	0.189198	0.205950
7	0.229133	0.329519
8	0.268576	0.359268
9	0.287889	0.384439
10	0.339280	0.487414
11	0.373486	0.567506
12	0.405237	0.585812
13	0.439280	0.663616
14	0.486743	0.691076
15	0.544845	0.704805
16	0.567430	0.718535
17	0.606056	0.739130
18	0.610311	0.771167
19	0.658265	0.787185
20	0.672504	0.800915
21	0.688216	0.828375
22	0.725368	0.837529
23	0.820131	0.864989
24	0.855319	0.878719
25	0.870213	0.881007
26	0.882160	0.901602
27	0.906383	0.931350
28	0.920458	0.947368
29	0.967430	0.967963
30	1.000000	1.000000

4.3 Implementation

The Artificial Neural Network (ANN) was configured with an input neuron derived from the normalized dataset and an output neuron for binary classification (0 or 1). The hidden layers used the Rectified Linear Unit (ReLU) activation function to scale outputs, while the output layer applied a linear function suitable for regression tasks. The optimal number of hidden neurons was determined experimentally, and the biases were set to match the sum of hidden neurons and the output neuron. The Decision Tree regressor model divided the data into branches using decision rules, recursively partitioning the feature space to predict the target variable. A random seed of 50 was specified for consistent results. In the Random Forest regressor model, the number of trees was set to 100, with a random seed of 50 for reproducibility. Ensemble predictions from these models were used as input for the Fireworks Algorithm (FWA), which refined the model parameters. The maximum training cycle was configured to randomly generate fireworks for the first FWA generation, and the optimal location was calculated to determine the amplitude and number of sparks. Gaussian sparks were generated and evaluated, selecting the best ones for the next generation until optimal weights and biases were achieved. The algorithm for training FWA is shown in Table 3.

Table 3. Algorithm for Training FWA

```
Begin
Randomly produce first generation of fireworks
For i=1 to maximum generation of fireworks do
Calculate error;
End for
Set optimal location
For j=1 to maximum training cycle do
    For i=1 to maximum generation of fireworks
        Evaluate amplitude;
        Evaluate number of sparks;
    End for
Produce sparks using Gaussian distribution;
Evaluate sparks;
Select sparks for next explosion;
Produce new generation of fireworks;
End for
Return values;
End
```

5. RESULTS

The experimental results are illustrated in Figures 4 to 7 to evaluate the performance of different predictive models based on normalized detected faults plotted against normalized execution time. The models assessed include Random Forest Regression, Decision Tree Regression, and Artificial Neural Networks (ANN), alongside an ensemble approach refined by Fireworks Algorithm (FWA). The results demonstrate strong predictive capabilities, as evidenced by the close alignment of the red (predicted) and blue (actual) points in the graphs. This alignment suggests that the models' predictions are generally accurate. However, a few gaps between actual and predicted faults reveal some minor deviations, indicating slight discrepancies in the model predictions.

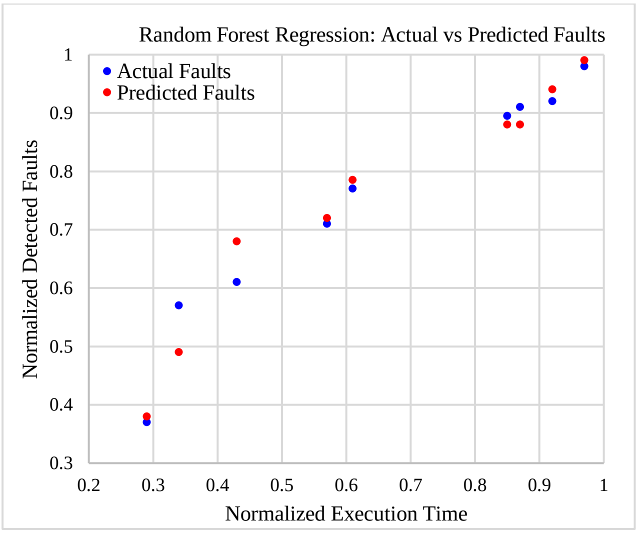

Fig 4: Random Forest Regression

The random forest regression model achieved a Mean Absolute Error (MAE) of 0.023, and a Mean Squared Error (MSE) of 0.0009575. The low MAE and MSE values reflect the model's accuracy in predicting fault counts, with the MAE showing that, on average, the predictions deviate by only 0.023 from the actual values. Meanwhile, the low MSE value indicates minimal squared differences between predicted and actual values, emphasizing the model's precision in capturing fault patterns. Together, these metrics confirm the Random Forest model's effectiveness in accurately predicting software faults based on normalized execution time, making it one of the most reliable models in this study.

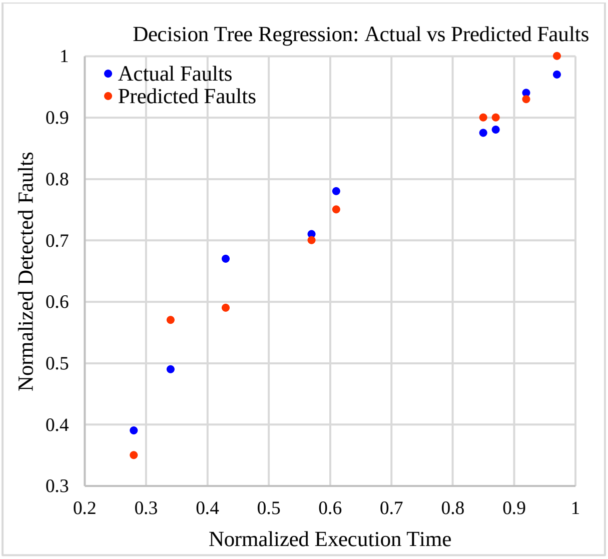

Fig 5: Decision Tree Regression

The decision tree regression model yielded a Mean Absolute Error (MAE) of 0.0355 and a Mean Squared Error (MSE) of 0.001838, indicating good predictive accuracy. The MAE suggests that predictions typically deviate from actual values by about 0.036 normalized units. Although the MSE is slightly higher than that of the Random Forest model, it remains low overall, highlighting the model's reliability in predicting software faults. While Decision Trees are known for their simplicity, this model's performance demonstrates its capability to handle the complexity of software fault prediction effectively.

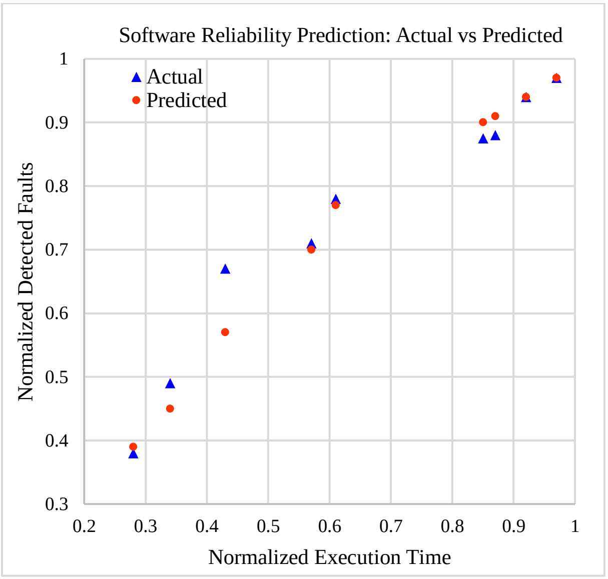

Fig 6: Artificial Neural Network

The artificial neural network (ANN) model achieved a Mean Absolute Error (MAE) of 0.0269 and a Mean Squared Error (MSE) of 0.001492, indicating strong predictive accuracy. The MAE value shows that, on average, predictions differ from actual values by only about 0.027 normalized fault units. The low MSE further confirms the model's ability to minimize significant errors in its predictions, reflecting the model's consistency. These metrics collectively suggest that the ANN model is effective in accurately forecasting software reliability, making it a reliable option for forecasting software faults, particularly in scenarios where complex relationships between variables need to be captured.

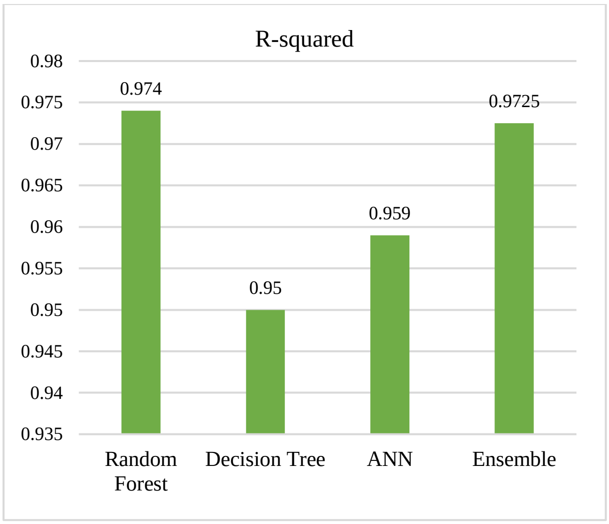

Fig 7: R-squared for Ensemble models

The R-squared (R^2) values for the random forest regression, decision tree regression, artificial neural network (ANN), and the ensemble model are 0.974, 0.950, 0.959, and 0.972, respectively. This measures how well the model explains the variance in the data. High R^2 values indicate that each model can explain a substantial proportion of the variability in detected faults based on execution time, with the random forest regression and the ensemble model showing particularly strong performance. An R^2 value of 0.974 for the random forest regression, for instance, means that 97.4% of the variance in detected faults is accounted for by the model, suggesting a high level of accuracy and fit. The slight differences among the models' R^2 values indicate variations in their predictive capabilities: while all models perform well, the random forest and ensemble models capture the data patterns more precisely. The decision tree model, with an R^2 of 0.950, performs slightly less accurately, which may reflect its simpler structure compared to the other methods. Overall, the high R^2 values across these models confirm that the chosen ensemble and individual methods are effective for fault prediction, and combining them may yield a model with balanced strengths, improving reliability and robustness in predictions.

The ensemble approach combines predictions from the Random Forest, Decision Tree, and ANN models to create a baseline prediction. These predictions then act as input for the Fireworks Algorithm (FWA) to further refine the ensemble results. The FWA further generates multiple sparks around the ensemble's baseline prediction while iteratively adjusting parameters to minimize prediction errors. After 50 generations, FWA achieved its best fitness, with an MSE of 0.0369 and an optimal Normalized Execution Time of 0.6319. The FWA's refinement significantly enhances the ensemble model's accuracy, reducing discrepancies and ensuring a high level of reliability in software fault prediction. By leveraging the strengths of individual models and further enhancing predictions through FWA, the ensemble approach demonstrates its potential to deliver balanced, precise, and robust predictions. This iterative refinement process shows the effectiveness of combining machine learning models with optimization algorithms for complex software reliability forecasting.

6. CONCLUSION

The results demonstrate that ensemble learning, combining Random Forest, Decision Tree, and Artificial Neural Network (ANN), provides a strong baseline for predicting software reliability by capturing patterns in execution time and fault detection with high accuracy. The ensemble model achieved an impressive R-squared (R^2) value of 0.972, reflecting that it explains 97.2% of the variance in detected faults. By applying the Fireworks Algorithm (FWA) to further refine these predictions, the model achieved even greater precision. FWA optimized the execution time parameters, yielding a best fitness with a Mean Squared Error (MSE) of 0.0369 at generation 50. The optimal Normalized Execution Time of 0.6319 (equivalent to approximately 311.15 in the original scale) indicates a minimal deviation from actual fault values, highlighting the effectiveness of this hybrid approach. Overall, integrating ensemble learning with FWA provides a powerful tool for accurate and reliable software fault prediction, making it highly valuable for real-time software reliability analysis.

Future research will explore integrating additional machine learning models into the ensemble, such as Support Vector Machines or Gradient Boosting, to further enhance predictive performance. Additionally, applying adaptive or dynamic tuning of FWA parameters will improve optimization efficiency, particularly for larger or more complex datasets. Finally, expanding the approach to analyze different types of software and fault scenarios will offer insights into its generalizability and adaptability across various domains.

7. REFERENCES

[1] Arora, M. and Choudhary, S. 2019. Software Reliability Prediction Using Neural Network.

[2] Benaddy, M. and Wakrim, M. 2012. Simulated Annealing Neural Network for Software Failure Prediction, International Journal of Software Engineering and Its Applications, Vol. 6, No. 4, pp. 35-46.

[3] Bisi, M. and Goyal, N. K. 2012. Software Reliability Prediction using Neural Network with Encoded Input, International Journal of Computer Applications (0975 – 8887), Volume 47, No. 22.

[4] Botchway, I., Alese, B. K. and Agangiba, W. A. 2021. "Evaluation of E-government Applications based on ISO/IEC 9126 Model", Annals Computer Science Series, Vol. 19, No. 1, pp. 26-36.

[5] Choudhary, A., Baghel, A., and Sangwan, O. 2017. Efficient parameter estimation of software reliability growth models using harmony search. IET Software 11(6):286–291.

[6] Dutta, K. K., Sunny, S. A., Victor, A., Nathu, A. G., Ayman Habib, M., and Parashar, D. 2020. Kannada alphabets recognition using decision tree and random forest models. Proceedings of the 3rd International Conference on Intelligent Sustainable Systems, ICISS 2020, 534–541.

[7] Guo, Y., Zhou, Y., Hu, X., and Cheng, W. 2019. Research on recommendation of insurance products based on random forest. Proceedings - 2019 International Conference on Machine Learning, Big Data and Business Intelligence, MLBDBI 2019, 308–311.

[8] Habtemariam, M. G., Mohapatra, S. K. and Seid, H. W. 2024. Software reliability prediction using ensemble learning with random hyperparameter optimization, Review of Computer Engineering Research, Vol. 11, No. 1, pp. 1-15.

[9] Habtemariam, M. G., Mohapatra, S. K., Seid, H. W., Prasad, S., Panigrahy, T. P. and Bal, P. K. 2024. Prediction and classification of software reliability using ensemble

learning, Journal of Integrated Science and Technology, Vol. 13, No. 2.

[10] Jijo, B. T., and Abdulazeez, A. M. (2021). Classification based on decision tree algorithm for machine learning. Evaluation, 6(7).

[11] Karunanithi, N., Malaiya, Y. K., and Whitley, D. (1991), Prediction of software reliability using neural networks. In Proceedings of the Second IEEE International Symposium on Software Reliability Engineering (pp. 124–130), 1991

[12] Lee, J., Sim, M. K. and Hong, J. S. 2024, Assessing Decision Tree Stability: A Comprehensive Method for Generating a Stable Decision Tree, IEEE Access, vol. 12, pp. 90061-90072

[13] Putra, P. H., Purba, B. and Dalimunthe, Y. A. 2023, Random Forest and decision tree algorithms for car price prediction, Jurnal Matematika Dan Ilmu Pengetahuan Alam LLDikti Wilayah (JUMPA), Vol. 3, No. 2, pp. 81-89.

[14] Saadah, S., and Salsabila, H. 2021. Prediksi Harga Ponsel Menggunakan Metode Random Forest. Jurnal Komputer Terapan, 7(1), 24–32.

[15] Sahu, K. and Srivastava, R. K. 2019. Revisiting Software Reliability

[16] Smarra, F., Di Girolamo, G. D., De Iuliis, V., Jain, A., Mangharam, R., and D'Innocenzo, A. 2020. Data-driven switching modeling for mpc using regression trees and random forests. Nonlinear Analysis: Hybrid Systems, 36, 100882.

[17] Sotarjua, L. M., and Santoso, D. B. 2022. Perbandingan Algoritma Knn, Decision Tree, Dan Random Forest Pada Data Imbalanced Class Untuk Klasifikasi Promosi Karyawan. Informatika Sains Dan, 7(2), 192–200.

[18] Su, Y.-S., and Huang, C.-Y. 2006. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. Journal of Systems and Software, 80(4), 606–615

[19] Zheng, J. 2009, Predicting software reliability with neural network ensembles, Expert Systems with Applications, Vol. 36, pp. 2116–2122

[20] Zhiwei, X., Kai, Z., Xin, X. and Juanjuan, H. 2020, A Firework Algorithm Based on Transfer Spark for Evolutionary Multitasking.