Journal of Trends in Computer Science and Smart Technology (ISSN: 2582-4104)
www.irojournals.com/tcsst/

Entangling Quantum Adversarial Network with Football Optimization for Software Defect Prediction

Suresh Yallamati[1], Shaheda Akthar[2]

[1]Research Scholar, [2]Lecturer in Computer Science, Government College for Women(A), Guntur, and Research Supervisor, Department of Computer Science, and Engineering, Acharya Nagarjuna University, A.P, India.

Email: [1]sureshyallamati@gmail.com, [1]researchwork660@gmail.com, [2]shahedaakthar76@gmail.com

Abstract

Predicting software defects is a critical component of software quality control. It is essential to plan for early defect detection and mitigation to enhance performance and reliability. Traditional machine learning and deep learning models often face challenges in managing missing values, extracting meaningful features, and effectively distinguishing between defective and non-defective software modules due to their reliance on linear classifiers and limited feature representation capabilities. To address these challenges, this study proposes an Entangling Quantum Generative Adversarial Network with Football Optimization Algorithm (EQGAN-FbOA) for efficient software defect prediction. The PROMISE dataset has been collected, and missing values are imputed during the pre-processing stage using Diffusion Models for Missing Value Imputation (DMVI), which employs a forward noising process to introduce controlled noise and a reverse denoising process to reconstruct the missing data. To enhance computational performance, a Spike-driven Transformer (S-DT) that incorporates a Leaky Integrate-and-Fire (LIF) spiking neuron is utilized for feature extraction. The EQGAN model improves defect prediction by generating quantum-enhanced feature representations. Additionally, the Football Optimization Algorithm (FbOA) is applied to balance exploration and exploitation through football-inspired search strategies, thereby preventing premature convergence and refining defect classification. Experimental findings from the PROMISE dataset demonstrate that the proposed method surpasses existing

DOI: https://doi.org/10.36548/jtcsst.2025.2.003
Received: 16.04.2025, received in revised form: 14.05.2025, accepted: 27.05.2025, published: 06.06.2025

approaches, achieving a software defect prediction accuracy of 99.8%, precision of 99.7%, recall of 99.6%, Matthews correlation coefficient (MCC) of 99.5%, and F-measure of 99.4%.

Keywords: Diffusion Models for Missing Value Imputation, Entangling Quantum Generative Adversarial Networks, Football Optimization Algorithm, Spike-Driven Transformer, Software Defect Prediction.

1. Introduction

Since the 1970s, the development of software defect prediction systems has emerged as one of the most active areas within software engineering. In today's environment, the economy, politics, social dynamics, and military capabilities are all heavily reliant on software. For highly complex and reliable software systems, ensuring software dependability is crucial. Software defects can lead to errors, crashes, and associated system issues [1]. With the rapid increase in software production, maintaining such reliability has become imperative [2].

A primary challenge in software defect prediction is the imbalance between minority-class and majority-class instances. This disparity leads prediction models to favor majority-class examples, ultimately diminishing overall prediction performance [3]. The projected volume of code in software is staggering; for example, Google's codebase encompasses over two billion lines, while a typical iPhone application contains between 10,000 and 15,000 lines [4]. Datasets used for software defect prediction often derive from static code measurements taken from issue log files of previous software versions [5]. These datasets consist of a collection of programs, data, or instructions utilized across various applications, including cybersecurity, to execute specific functions [6]. By analyzing a software module's source code or development process, programmers can proactively identify defects and concentrate on problematic modules to enhance software quality [7].

Hyperparameter tuning significantly influences classifier performance, with studies indicating that this technique can lead to improved prediction outcomes [8]. The Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU) models are among the most prominent deep learning architectures designed to address issues related to gradient vanishing and long-term dependencies. These models exhibit strong prediction performance in software defect prediction due to their ability to identify longer time-series data sequences [9]. Utilizing this type of data enhances feature semantics, enabling software defect prediction models to

more effectively detect malfunctioning modules. However, the unstructured nature of natural language data presents challenges in integrating natural language texts with programming language code to fully understand semantic aspects [10].

When labeled target instances are available, they can be employed for instance-based transfer learning without requiring similarity metrics between source and target instances. Additionally, variations in conditional distribution have been utilized for multi-source applications in the second stage of the process [11]. Although no single strategy is universally effective, machine learning models must adapt their hyperparameters for each classification challenge to optimize performance [12]. Insufficient attention has been paid to the application of machine learning techniques for predicting the number of defects in a software module [13].

Improving software defect prediction technology can reduce Research and Development (R&D) costs and contribute to the creation of more reliable software systems. Both inexperienced engineers and inadequate software development processes are significant contributors to software defects [14]. As an instance-based transfer learning approach, it can leverage labeled target instances when available, without necessitating similarity metrics between source and target instances [15].

1.1 Novelty and Contribution

- The PROMISE dataset undergoes data cleaning to handle missing values. DMVI applies a forward noising process and reverse denoising to reconstruct the missing data.
- S-DT with LIF spiking neurons extracts relevant software defect features. This enhances computational efficiency and improves feature representation.
- EQGAN generates quantum-enhanced feature representations for defect classification. Fidelity-based discrimination ensures accurate distinction between defective and non-defective modules.
- EQGAN optimizes weight parameters using the Football Optimization Algorithm (FbOA). This ensures robust classification and enhances predictive performance for accurate SDP.

The format of this report is as follows: The literature is examined in Section 2, and the recommended plan of action is presented in Section 3. Section 4 displays the results and comments. In Section 5, the Conclusion and suggestions for further study are provided.

2. Literature Survey

In 2024 Abdul Waheed Dar and Sheikh Umar Farooq [16] introduced Software Defect Prediction (SDP), which finds faulty modules and maximizes testing resources, is necessary to raise the caliber of software. However, class imbalance and overlap reduce prediction accuracy in SDP datasets.This work proposes a four-stage pipeline consisting of Extreme Gradient Boosting (XGBoost), an under-sampling strategy, and a class overlap reduction technique to improve prediction performance in an ensemble SDP model. After being evaluated on sixteen imbalanced SDP datasets and contrasted with ten cutting-edge methods, the model successfully manages overlap and class imbalance problems. It lowers development costs, improves error detection, and increases predictive accuracy. Though computationally demanding, the method greatly enhances classification performance. In 2023 Iqra Mehmood et.al [17] developed to increase the accuracy of machine learning classifiers, this research uses feature selection to improve Software Defect Prediction (SDP). Utilizing the Waikato Environment for Knowledge Analysis (WEKA) application, the suggested method picks features utilizing classifiers such Random Forest, Logistic Regression, Multilayer Perceptron, Bayesian Network, J48, Lazy IBK, Support Vector Machine, Neural Networks, and Decision Stump. The objective is to improve the defect prediction accuracy in the PROMISE dataset using five publicly available National Aeronautics and Space Administration (NASA) datasets: CM1, JM1, KC2, KC1, and PC1. The Minitab software is used for statistical analysis. The approach's advantage is that it improves prediction accuracy compared to models without feature selection; however, its disadvantage is that feature selection increases computing complexity. In 2023 Nasraldeen Alnor Adam Khleel and Karoly Nehez [18] established SDP improves software quality by identifying troublesome components using previous defect data. This study employs a Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU) in conjunction with the Synthetic Minority Oversampling Technique and the Tomek link (SMOTE Tomek) to address the issue of class imbalance in SDP. Improved prediction performance is demonstrated by experiments using PROMISE repository datasets, with GRU

attaining a 24% better Area CNN receives a Receiver Operating Characteristic Curve (AUC) of 19%. Compared to existing SDP models, the proposed approach enhances defect prediction in terms of mean square error (MSE), accuracy, precision, recall, Matthew's Correlation Coefficient (MCC) and F-measure. In 2022 Pravali Manchalaa and Manjubala Bisi [19] suggested the purpose of SFP is to identify between defective and non-defective modules; model performance is impacted by class imbalance. Weighted Average Centroid-based Imbalance Learning Approach (WACIL) is proposed as a synthetic oversampling method that enhances diversity and decreases noise.

WACIL filters noise and generates pseudo-data from borderline events using a weighted average centroid algorithm. Based on experiments on 24 PROMISE and National Aeronautics and Space Administration (NASA) datasets, the best methods in terms of False Omission Rate (FOR), F-measure, and Area Under the Receiver Operating Characteristic Curve (AUC) are K-Nearest Neighbours (KNN), Logistic Regression (LR), Naïve Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), and Deep Neural Network (DNN). With statistical evidence to support its effectiveness, WACIL is a good choice for addressing class imbalance in SFP. In 2022 Li-qiong Chen *et.al* [20] evaluated the software testing ensures high-quality software, and Software Defect Prediction (SDP) aids in efficient defect detection. In order to improve prediction accuracy and resource allocation, this paper suggests an SDP framework that makes use of heterogeneous feature selection and nested stacking. A Nested-Stacking classifier, feature selection, dataset pre-processing, and performance assessment are all included in the system. Region The Receiver Operating Characteristic Curve (AUC) and F1-score are used in experiments to evaluate classification performance on the Kamei and PROMISE datasets. By displaying improved accuracy in Within-Project defect Prediction (WPDP) and Cross-Project Defect Prediction (CPDP), the model surpasses baseline approaches and increases the effectiveness of software fault classification.

In 2023 Iqra Mehmood et.al [21] demonstrated the software engineering, defect prediction is essential for locating source code defects prior to testing. It employs a variety of feature selection and machine learning techniques to increase prediction accuracy, including Random Forest, Logistic Regression, Multilayer Perceptron, Bayesian Network, J48, Lazy IBK, Support Vector Machine, Neural Networks, and Decision Stump. Feature selection (WFS) improves accuracy over WOFS, according to experiments conducted with Minitab and the Waikato Environment for Knowledge Analysis (WEKA) on the publicly available National

Aeronautics and Space Administration (NASA) datasets CM1, JM1, KC2, KC1, and PC1. In 2024 MISBAH ALI et.al [22] suggested detecting and delivering only faulty modules for testing, software defect prediction improves software quality and lowers testing expenses. This study combines Random Forest, Support Vector Machines, Naïve Bayes, and iteratively updated artificial neural networks to produce intelligent ensemble-based software defect prediction model.On seven NASA Metrics Data Program datasets, a voting ensemble outperforms 20 techniques in terms of accuracy by aggregating their predictions. In 2023 Shabib Aftab et.al [23] determined an intelligent cloud-based Software Defect Prediction (SDP) system, data and decision-level machine learning techniques are merged. Its prediction approach consists of two steps: Naïve Bayes, Artificial Neural Networks (ANN), and Decision Trees (DT) are used first, followed by fuzzy logic-based rules that combine classifier accuracy. The tests performed better than ensemble approaches and basic classifiers, achieving 91.05% accuracy on the National Aeronautics and Space Administration (NASA) CM1, MW1, PC1, PC3, and PC4 datasets.In 2022, Mutasem Shabeb Alkhasawneh [24] demonstrated that millions of people worldwide are impacted by software flaws, which result in large financial losses. The proposed method integrates feature selection and classification using a Radial Basis Function Neural Network (RBF) with a correlation-based methodology. The National Aeronautics and Space Administration (NASA) provided fourteen datasets for the model's evaluation using K-fold cross-validation. Performance was evaluated using F-measure, accuracy, precision, and recall. The results confirmed that the model was effective in increasing software dependability, with superior defect prediction compared to competing methods, particularly for the datasets CM1, KC4, MC1, PC1, PC2, PC3, PC4, and PC5. In 2022, K. Thirumoorthy and J. Jerold John Britto [25] noted that software defects can lead to severe economic consequences, making early prediction essential. Software modules that are prone to errors are categorized in this study using the Elitist Self-Adaptive Multi-Population Social Mimic Optimization (ESAMP-SMO) technique as part of a clustering-based Software Defect Prediction (SDP) method. While optimizing the fault prediction rate, the objective function reduces intra-cluster distance. The CM1, JM1, and KC1 datasets from the National Aeronautics and Space Administration (NASA) are used in experiments to verify better performance than current techniques. Table 1 shows the comparative analysis of the literature survey.

Table 1. Comparative Analysis of Literature Survey

Reference	Method	Advantages	Disadvantages
[16]	XGBoost	The ensemble SDP model improves accuracy and reduces costs by balancing data.	The method requires high computation, increasing processing time and resource usage.
[17]	Bayesian Net	Feature selection enhances defect prediction accuracy in SDP models.	Feature selection uses extra processing power and adds complexity to the algorithm.
[18]	CNN	The method successfully corrects imbalance and increases the accuracy of defect prediction.	CNN and GRU increase model complexity and computational cost.
[19]	WACIL	Improves data diversity, reduces noise in SFP	Computational complexity, potential data overfitting issues
[20]	Nested-stacking classifier	Enhances prediction accuracy, improves resource allocation	Increased computational cost, complex framework implementation
[21]	WEKA	Increases precision by employing feature selection (WFS).	Few datasets were used for assessment.
[22]	VESDP	Improves prediction accuracy by merging several classifiers.	computationally expensive due to iteration

[23]	FSDPS	Achieves high accuracy, surpassing base classifiers.	Increased complexity as a result of integrating fuzzy logic
[24]	RBFNN	It improves the ability to predict defects across various datasets	Requires a large amount of processing power.
[25]	ESAMP-SMO	Improved fault prediction and decreased intra-cluster distance	Improved fault prediction and decreased intra-cluster distance

2.1 Summary

The comparative analysis discusses a number of SDP strategies and their benefits and drawbacks. XGBoost uses a lot of processing resources even though it lowers expenses and increases accuracy. Bayesian Net increases complexity while improving prediction through feature selection. CNN corrects the imbalance while increasing computing costs. Despite its computational issues, the nested-stacking classifier maximizes resource use, while WACIL encourages data diversity. Despite their complexity and dataset limitations, methods like WEKA, VESDP, and FSDPS improve accuracy. RBFNN enhances defect prediction but requires a significant amount of processing power, whereas ESAMP-SMO enhances fault detection and reduces intra-cluster distance.

2.2 Problem Statement

Software Defect Prediction (SDP), it detects issues early in the development process and is essential for ensuring software reliability. However, problems including data imbalance, noise, and computational complexity make it difficult to achieve high prediction accuracy. Many current models suffer from overfitting, high processing costs, and poor generalizability. Furthermore, striking a balance between efficiency and precision is still a major issue. An enhanced SDP technique that improves fault prediction while maximizing computational resources and preserving model scalability is thus required.

3. Proposed Methodology

The suggested Entangling Quantum Generative Adversarial Network with Football Optimization Algorithm (EQGAN-FbOA) is designed for effective software defect prediction. The PROMISE dataset, which comprises 1,125 examples divided into 80% training (900 instances) and 20% testing (225 instances), is first obtained. These instances are classified as Defective (985) and Non-defective (140). Diffusion Models for Missing Value Imputation (DMFMVI) are used for pre-processing. A Spike-driven Transformer (S-DT) is utilized for feature extraction to improve computational performance. For defect prediction, EQGAN generates quantum-enhanced feature representations and performs joint measurements on both real and generated defect-prone module representations, ensuring accurate defect classification through fidelity computation. To further optimize prediction performance, the FbOA enhances the process by simulating football strategies, balancing exploration and exploitation for improved defect detection. This combined approach strengthens software quality assessment by integrating quantum generative learning with football-inspired optimization techniques, leading to more precise and efficient software defect prediction (SDP).. Figure 1 shows the workflow of the proposed EQGAN-FbOA.

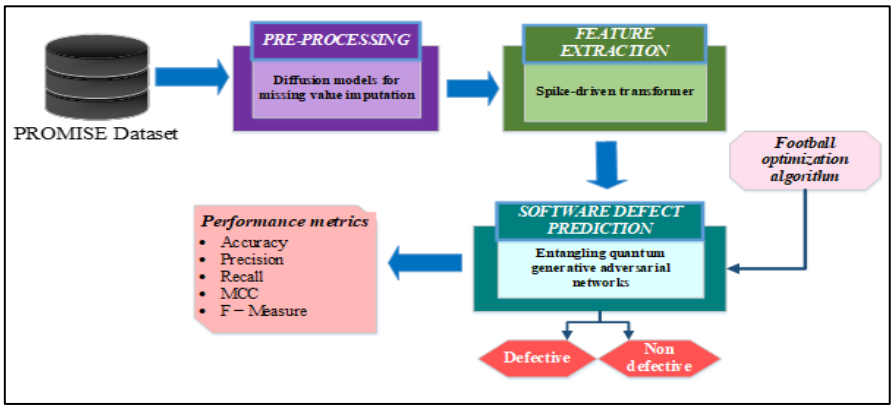

Figure 1. Block Diagram of the Proposed Methodology

3.1 Data Acquisition

To evaluate software defect prediction, the PROMISE dataset is utilized, containing two main classes: Defective and Non-defective, which are used to assess the effectiveness of the suggested approach. There are 1,125 cases in the dataset, 985 of which are classified as defective and 140 as non-defective. The dataset is divided into 20% testing (225 instances) and 80% training (900 instances) for model evaluation. Name, version, amount of code files,

and defect rate are among the crucial project details it contains. In order to examine software defect prediction, 20 static metric features are also extracted.

3.2 Pre-Processing Using Diffusion Models for Missing Value Imputation

The DMFMVI [26] aims to improve model performance. Pre-processing is an essential step in data analysis that enhances data quality by reducing noise and increasing clarity. Although the PROMISE dataset is frequently used to predict software defects, model performance may be impacted by missing values in software measurements. Through a two-step procedure called forward noising and reverse denoising, diffusion models provide an efficient solution for missing value imputation. The forward process fixes the observed portion of the data while gradually adding Gaussian noise to the unobserved portion. The opposite procedure then learns the conditional distribution and iteratively reconstructs the missing data. For missing value imputation in the PROMISE dataset, pre-processing involves standardizing numerical attributes and encoding categorical variables. A diffusion-based imputation process iteratively refines missing values using learned conditional distributions. Post-imputation, decoding strategies such as probability-based selection, thresholding, and nearest-neighbour methods ensure data integrity. This approach enhances defect prediction accuracy and software quality assessment. An input dataset is separated into two parts: the unobserved y_{ua} (missing values to be predicted) and the observed y_{b0} (known values). Equation (1) explains the conditional distribution is approximated by the model.

$$h\theta\left(y_{u-1}^{ua}\middle|y_{u}^{ua},y_{0}^{b0}\right)=M\left(y_{u-1}^{ua};\mu_{\theta}\left(y_{u}^{ua},u\middle|y_{0}^{b0}\right),\sigma K\right) \tag{1}$$

where the conditional expectancies based on observed and missing data are represented by the function $h\theta$. The variable y_u^{ua} represents the missing values, while y_0^{b0} defines the baseline observed data, serving as a reference for imputation. The transformation procedure that improves missing value estimates is represented by the mapping function $M(.)$. The variance term σK indicates the uncertainty in the imputation process, guaranteeing robustness in estimation, whereas the mean estimation function determines the expected value of missing data. This approach aims toenhance data quality and improve model performance by effectively imputing missing values in the PROMISE dataset. By leveraging diffusion models, the approach ensures robust estimation, preserving data integrity for accurate defect prediction and

software quality assessment. Following pre-processing, feature extraction is covered in the next section.

3.3 Feature Extraction Using Spike-Driven Transformer(S-DT)

The process of converting raw data into a collection of features that a spike-driven transformer can use efficiently is known as feature extraction. Using sparse addition operations to improve computing performance, the S-DT [27] combines the architecture of the Transformer with the spike-driven paradigm. An essential component of this idea is the computationally effective Leaky Integrate-and-Fire (LIF) neuron. Equations (2), (3), and (4) describe the functioning of the LIF neuron.

$$E[s] = G[s-1] + Y[s], \tag{2}$$

$$V[s] = Gnb(E[s] - v_{th}), \tag{3}$$

$$G[s] = N_{reset} V[s] + (\beta V[s])\,(1 - V[s]) \tag{4}$$

Where, s denotes the time step and $E[s]$ represents the membrane potential that results from coupling the temporal input $G[s-1]$ with the spatial input information $Y[s]$, where $Y[s]$ can be acquired using operators like self-attention, MLP, and Conv. A higher membrane potential than the threshold v_{th} will cause the neuron to produce a spike.; else, it won't. Consequently, there is only 1 or 0 in the spatial output tensor $V[s]$. The Heaviside step function $Gnb(y)$ satisfies the condition $Gnb(y) = 1$ when $y \geq 0$ and $Gnb(y) = 0$ otherwise. Whereas N_{reset} represents the reset potential that is set upon the activation of the output spike, $G[s]$ represents the temporal output. The membrane potential $E[s]$ will decay to $G[s]$ if the spiking neuron does not fire, with $\beta < 1$ as the decay factor. Based on spike activity, SPS dynamically divides input patches. Spike-driven patch splitting is explained in Equation (5).

$$v = PSM(J) \tag{5}$$

where, $v = PSM(J)$ is represented by the input stimulus or activation, v is the calculated output, and PSM stands for the Patch Splitting Mechanism. This function

dynamically segments input patches according to spike activity in order to increase computing efficiency while processing J utilize the spike-driven patch splitting. Equation (6) defines the spike-driven self-attention to extract local-global dependencies.

$$H'_m = SDSA(V_{m-1}) + H_{m-1} \tag{6}$$

where, the feature representation update is represented by $H'_m = SDSA(V_{m-1}) + H_{m-1}$. here, the updated feature map at layer m is represented as H'_m, $SDSA$. The Spike-Driven Self-Attention mechanism is shown by $SDSA$, the input from the previous layer, $m-1$, is represented by V_{m-1}, and the feature representation from the previous layer is represented by H_{m-1}. By combining spike-driven self-attention with previous feature information, this formulation improves feature extraction. The final class result is expressed in Equation (7).

$$X = CH(GAP(V_K)) \tag{7}$$

where, CH is a channel-wise modification or compression function, V_K is the input feature map, X is the final output representation, and GAP is the Global Average Pooling operation done to it. While maintaining crucial channel-wise information, this formulation helps to the reduction of spatial dimensions. The integration of the spike-driven transformer improves feature extraction efficiency, optimizing local-global feature representation and reducing computational complexity. After feature extraction, the next step is prediction.

3.4 Prediction Using Entangling Quantum Generative Adversarial Networks (EQGAN)

In EQGAN for software defect prediction, quantum-enhanced feature representations are generated to capture complex patterns in software data. The discriminator evaluates entangled quantum states, refining defect detection accuracy and optimizing software quality assessment. Using the PROMISE dataset, EQGAN [28] present a unique minimax optimization method for predicting software defects. The EQGAN performs joint measurement on both the created defect-prone module representations and the real software defect data, in contrast to standard GANs where the discriminator assesses generated and real data independently. Software defect prediction models are optimally evaluated which facilitates this measurement. For complex defect classification, traditional quantum generative models may not be the best option because they depend on a linear function of input states. To improve defect prediction

in software systems, EQ-GAN, on the other hand, uses a parameterized fidelity-based discriminator to differentiate between produced and genuine defect patterns. The reliability of the measurement function is shown in Equation (8).

$$E_{\sigma}^{fid}\left(\rho(\theta_k)\right)=\left(Hz\sqrt{\sigma^{1/2}\rho(\theta_k)\sigma^{1/2}}\right)^2 \tag{8}$$

Where, the trace function Hz determines the relationship between $\rho(\theta_k)$ and the reference covariance matrix σ, ensuring accurate defect pattern differentiation in EQ-GAN. The equation represents the fidelity-based expectation E_{σ}^{fid} of the quantum state $\rho(\theta_k)$.A minimax cost function is explained in Equation (9).

$$\min_{\theta_k}\max_{\theta_h}G(\theta_k,\theta_h)=\min_{\theta_k}\max_{\theta_h}\left[1-E_{\sigma}(\theta_h,\rho(\theta_k))\right] \tag{9}$$

where, the parameters of the generator are represented by θ_k.The parameters of the discriminator are denoted as θ_h. The discriminator and generator are balanced by the goal function is $G(\theta_k,\theta_h)$. The trace function norm that measures similarity is indicated by E_{σ} $\rho(\theta_k)$ represents the generated quantum state. The discriminator as follows in Equation (10).

$$\begin{aligned}UH(\theta_e)U|0\rangle_b|\varphi\rangle|\zeta\rangle=&\frac{j\sin\theta_h}{2}|1\rangle_b\left[|\zeta\rangle|\psi\rangle-|\psi\rangle|\zeta\rangle\right]\\&+\frac{1}{2}|0\rangle_b\left[\left(f^{-j\theta_h}+\cos\theta_h\right)|\psi\rangle|\zeta\rangle-j\sin\theta_h|\zeta\rangle|\psi\rangle\right]\end{aligned} \tag{10}$$

Where a $UH(\theta_e)$ unitary transformation is applied to the quantum state. The basis states of the quantum system are $|0\rangle_b$ and$|1\rangle_b$. $|\varphi\rangle|\zeta\rangle$ represents the quantum state input. j denotes the unit of imagination. Trigonometric terms that determine state evolution are $\sin\theta_h$ and $\cos\theta_h$. $f^{-j\theta_h}$ represents the transformation's phase factor. Equation (11) is used to determine the probability of measuring state $|0\rangle$.

$$E_{\sigma}(\theta_h,\rho(\theta_k))=\frac{1}{2}\left[1+\cos^2\theta_h+\sin^2\theta_h E_{\sigma}^{fid}(\rho(\theta_k))\right] \tag{11}$$

where, the expected value is represented as $E_\sigma(\theta_h, \rho(\theta_k))$ depending on the parameters θ_h and the quantum state $\rho(\theta_k)$. A scaling factor of $\frac{1}{2}$ is included in the equation to guarantee normality. The weight distribution of the various components is controlled by the parameters of $\cos^2\theta_h$ *and* $\sin^2\theta_h$. The reliability of the measurement linked to the quantum state is also represented by the fidelity-based expectation function is $E_\sigma^{fid}(\rho(\theta_k))$.By using quantum-enhanced generative adversarial training to precisely model defect-prone modules, the EQ-GAN technique improves software defect prediction. It improves software quality evaluation by better differentiating between created and real fault patterns through the use of fidelity-based quantum measurements. In the next section, the EQGAN's weight parameters are E, H optimized using the Football optimization algorithm.

3.5 Football Optimization Algorithm (FbOA)

The FbOA[29] is a metaheuristic optimization technique inspired by football strategies, including passing, positioning, and teamwork. It models the optimization process as a football game, where each agent (player) navigates the search space using short passes (local search), lob passes (intermediate search), and through-ball passes (global search). These techniques help maintain a balance between exploration and exploitation, preventing premature convergence to local optima while ensuring efficient search performance. FbOA is applied to software defect prediction by simulating real-time decision-making and adaptive positioning to identify defect-prone modules.

Step 1: Initialization

Football players adjust their moves, pass strategically, and position themselves to create scoring opportunities; these tactics are the inspiration behind FbOA. In order to ensure dynamic flexibility and effective solution discovery, it emulates this optimization strategy by using through-ball passes for global search, lob passes for intermediate exploration, and short passes for local search.

Step 2: Fitness Function

The fitness function measures how well defects are identified in order to assess the quality of solutions in FbOA. It assesses how effectively a player or agent moves up the search

space. Because the function is based on velocity and force updates, agents are guaranteed to efficiently investigate promising defect patterns. The fitness function evaluates the quality of each software defect prediction model by measuring the accuracy of each solution's classification of fault-prone modules. Equation (12) provides an explanation of the function's definition.

$$Fitnessfunction = Min(E,H)Max(Accuracy) \tag{12}$$

where, the fitness function maximizes accuracy to optimize the optimal solution by choosing the minimum of weights E and H.

Step 3: Exploration

The FbOA's exploration performance is predicated on the idea that a player will always execute his pass with a specific velocity. To avoid becoming stuck in the local optimum, the search space must be explored while current regions are exploited. The dynamic process is explained in Equation (13).

$$Fb(G(s+1)) = j \tag{13}$$

where, Fb represents the evaluation function for input. $G(s+1)$ denotes the transformation based on s represented by a function G applied to $s+1$. s Stands for a state index or iteration. j Value that results from applying Fb to $G(s+1)$.

Step 4: Football Velocity

FbOA dynamically adjusts the movement speed of agents, ensuring a balance between exploration and exploitation. It incorporates external forces, acceleration factors, and trigonometric modulation to enhance search efficiency in software defect prediction. The formula shows each player's velocity is explained in Equation (14).

$$K_m = F_{\max}\left(a_y \cdot b_j\left[F_{ext} - F_{\min}\right] + s \cdot a_x \cdot b_i\left[F_{best} - F_{\min}\right] \times \cos\left(\frac{\pi}{iteration}\right)\right) \tag{14}$$

Where, the number of factors determines the player's velocity at iteration m, which is represented as K_m. A velocity upper limit is established by the highest force applied, $F_{\max}$, which guarantees regulated movement inside the search space. The influence of various

directions on the agent's movement through the space is determined by the coefficients b x and a_y. The acceleration factors determine the b_i and b_j are essential for modifying speed since they can either enhance or reduce the agent's motion. External influences, which take into consideration extra forces influencing the search process, are represented by F_{ext}. The lower bound of velocity is indicated by the $F_{\min}$. The optimal force discovered thus far in the optimization process is denoted by F_{best}. s is a random variable that increases exploration and adds variability. $\cos\left(\frac{\pi}{iteration}\right)$ represents the speed is adjusted based on the number of repetitions to maintain a balance between exploration and exploitation.

Step 5: Update for Best Force

The FbOA Update for the Best Force equation dynamically modifies the best-found solution to further hone the search. Through repeated updates, it strikes a balance between exploration and exploitation, ensuring an individualized yet flexible optimization process. The optimal force is updated using Equation (15).

$$F_{best} = \frac{1}{P}\sum_{m=0}^{L}\left(\frac{E_{\max}^{m^2}}{(2n+1)^2}\right) \tag{15}$$

where, the most effective force in the current optimization context is denoted by F_{best}. L is an exponential variable that balances exploration and exploitation by rising from 0 to 1 over the course of the iterations. $(2n+1)^2$ is a normalization term to control the update rate. F_{best} indicates the power raise.

Step 6: Exploitation

The FbOA in exploitation focuses on improving solutions within designated search spaces, concentrating on lucrative areas during exploration. The update rules for agent location changes during the exploitation phase is expressed in Equation (16).

$$Fb(G(s+1)) = F_j + v_3 \cdot Fb(G(s)) + D \cdot \sin\left(\frac{\pi}{Iteration}\right) \tag{16}$$

where, the modified solution at iteration $s+1$ is denoted by $Fb(G(s+1))$. Fi represents the current position. A control parameter called v_3 modifies how the current and new positions are balanced. At iteration s, the current location or state is denoted by $Fb(G(s))$. D is an exponential variable that rises as the iterations continue. The $\sin\left(\frac{\pi}{Iteration}\right)$ speeds up convergence to the ideal solution by introducing a sinusoidal modulation.

Step 7: Termination

Upon reaching a predetermined maximum number of iterations, achieving convergence where the difference between the best solutions in successive iterations drops below a threshold or reaching solution stability. where the fitness function value stabilizes, signifying no further improvement FbOA comes to an end. Through the combination of adaptive search mechanisms and football-inspired techniques, FbOA improves software defect prediction by precisely identifying modules that are prone to defects while preserving strong search efficiency. The Football Optimization Algorithm (FbOA) maximizes classification accuracy and avoids premature convergence in order to enhance software defect prediction. It improves flaw identification by establishing a balance between search space exploration and exploitation. Figure 2 shows the Football Optimization Algorithm.

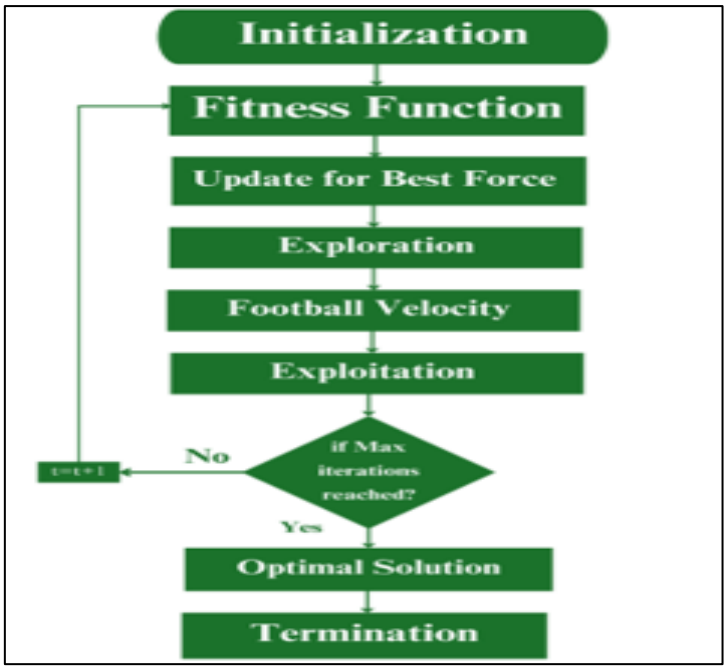

Figure 2. Flow Chart for Football Optimization Algorithm

4. Results and Discussion

The proposed method utilizes Python 3.7.1 to process software defect data on Windows 10. Training and testing have been conducted using the PROMISE dataset. The EQGAN neural network is employed for software defect prediction, with the model parameters detailed in Table 2.

Table 2. Implementation Parameters

Parameters	Values
Windows	10
Programming language	Python version 3.7.1
Neural Network	EQGAN
Optimization	Football Optimization Algorithm
Dataset	PROMISE

4.1 PROMISE Dataset Description

To evaluate the accuracy of the proposed strategy, six open-source Java programs were selected from the PROMISE dataset. All six projects have publicly available source codes and PROMISE [17] data. These projects, which comprise applications such as XML parsers, data transport adapters, and text search engine libraries, provide traditional static metrics for each Java file. The experimental datasets consistof projects with different sizes and fault rates to ensure that the evaluation results are generalizable. The defect rate ranges from 2.23% to 92.19%, while the number of incidents across the six projects ranges from 205 to 965. Key details about the selected projects are shown in Table 3, including the project name, version, number of instances, and defect rate, or the percentage of problematic instances.

Table 3. Description of the PROMISE Dataset

Project version	Project name	Defect rate (%)	#of instances
1.6	camel	19.48	965
1.7	ant	22.28	745
2.0	ivy	11.36	352
1.2	Log4j	92.19	205
1.4	xerces	74.31	588
4.3	Jedit	2.23	492

4.2 Performance Metrics

Table 3 explains the performance metrics, including F-Measure, Recall, Precision, MCC (Matthews Correlation Coefficient), and Accuracy.

True Positive (A): Accurately determinesthat a software module has become defective.

True Negative (B): Identifies a software module as non-defective with accuracy.

False positive (C): Incorrectly identifies a software module that isn't defective as one.

False Negative (D): Fails to identify a malfunctioning software module and incorrectly labels it as non-defective.

Table 4. Performance Metrics

Performance Metrics	Formula
Accuracy	$(A+B)/(A+C+D+B)$
Precision	$\frac{A}{A+C}$
Recall	$A/(A+D)$

MCC	$A*B-C*D/\sqrt{(A+C)*(A+D)*(B+C)*(B+D)}$
F-Measure	$(2*\text{Re}call*\text{Pr}ecision)/(\text{Re}call+precision)$

4.3 Performance Analysis of the Proposed Model

The suggested EQGAN-FbOA approach improves software fault prediction by systematically processing the PROMISE dataset through steps of feature extraction, pre-processing, and prediction. SDP accuracy has enhanced when the model has been validated for performance analysis utilizing machine learning and optimization methodologies.

Table 5. Testing Outcome of Proposed Model for PROMISE Dataset

Module ID	**Input Data**		**Pre-processing**		**Feature Extraction**	**Prediction**
1	Ioc=200	v(g)=10	Ioc=0.8	V(g)=0.4	v(g)=0.4	Defective
	iv(g)=5	n=50	Iv(g)=0.2	n=0.6	iv(g)=0.2	
	V=100	d=20	v=0.7	d=0.4	n=0.6	
	i=5	e=500	i=0.3	e=0.8	e=0.8	
	b=2	t=10	b=0.2	t=0.4	d=0.4	
2	Ioc=100	v(g)=5	Ioc=0.5	v(g)=0.2	v(g)=0.2	Non-Defective
	iv(g)=2	n=25	iv(g)=0.1	n=0.5	iv(g)=0.1	
	v=50	d=10	v=0.5	d=0.3	n=0.5	
	i=2.5	e=250	i=0.2	e=0.7	e=0.7	
	b=1	t=5	b=0.1	t=0.3	d=0.3	

The EQGAN-FbOA model's testing results for software defect prediction using the PROMISE dataset are shown in Table 5. It describes the various module IDs, the input data that goes with them, the pre-processing procedures, the features that are extracted, and the final

forecasts. The pre-processing step normalizes several software metrics, including cyclomatic complexity (v(g)), lines of code (loc), effort (e), and number of decisions (d), to guarantee effective feature representation. These metrics are further refined for better defect categorization through feature extraction. Using learned patterns, the model successfully separates modules that are defective from those that are not. The outcomes demonstrate how well the suggested method predicts software flaws, making it a dependable tool for evaluating software quality and identifying errors.

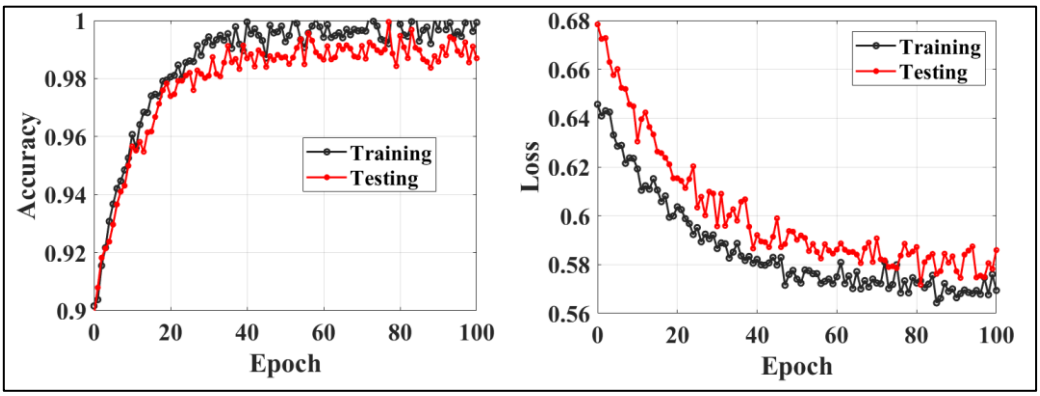

Figure 3. Proposed (a) Accuracy and (b) Loss for PROMISE Dataset

The model's accuracy (left) and loss (right) for training and testing data across 100 training epochs are shown in Figure 3. The loss plot exhibits a continuous drop, whereas the accuracy plot shows a gradual improvement, approaching near-optimal performance. Overall, the model generalizes effectively, although the testing curves' modest oscillations point to some overfitting. By assessing model performance, these learning curves guarantee efficient training and reduce the possibility of overfitting.

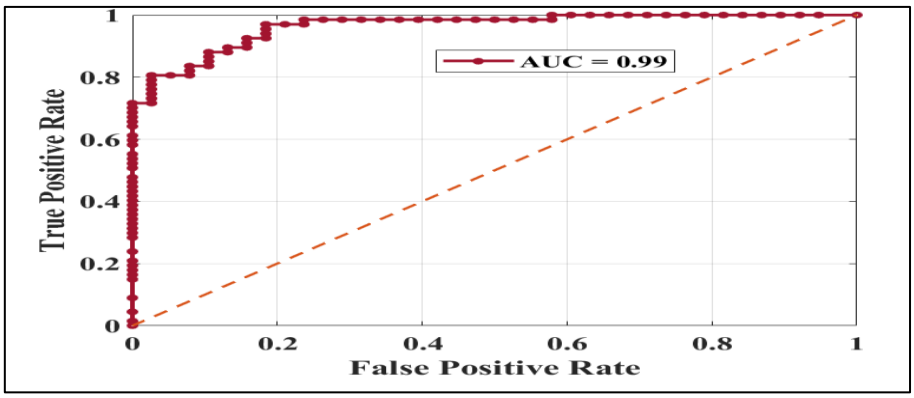

Figure 4. Proposed ROC Curves for PROMISE Dataset

Plotting the True Positive Rate (TPR) versus the False Positive Rate (FPR) after applying the Receiver Operating Characteristic (ROC) curve to the PROMISE dataset in Figure 4 shows the prediction efficiency of the proposed approach. The outstanding predictive potential of the model is indicated by its Area Under the Curve (AUC) of 0.99. The curve displays excellent sensitivity and specificity because it is near the top-left corner. Since the model's curve is significantly above the diagonal orange dashed line, which represents random guessing, the classifier performs significantly better than chance.

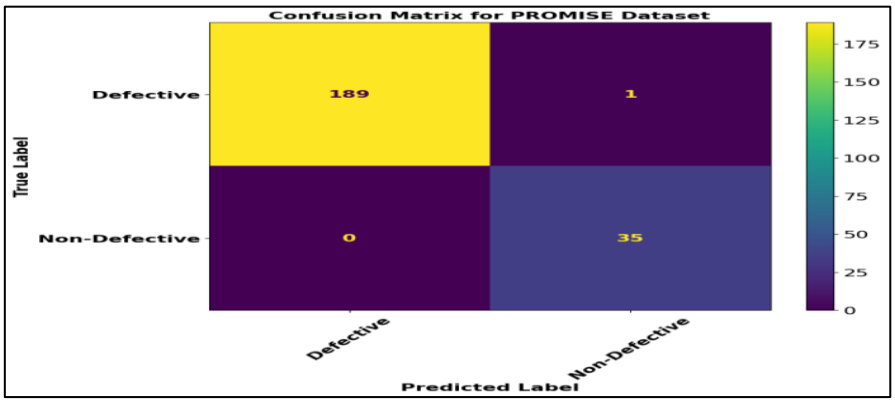

Figure 5. Proposed Confusion Matrix for PROMISE Dataset

Figure 5 shows the performance of the proposed model for the PROMISE dataset. Four important metrics are included: one false negative (items that were mistakenly classified as non-defective), 35 true negatives (items that were correctly identified as non-defective), 189 true positives (items that were correctly predicted to be faulty), and 0 false positives (no non-defective items that were mistakenly classified as defective). The high number of correct predictions and low misclassification rate demonstrate how well the model distinguishes between defective and non-defective objects.

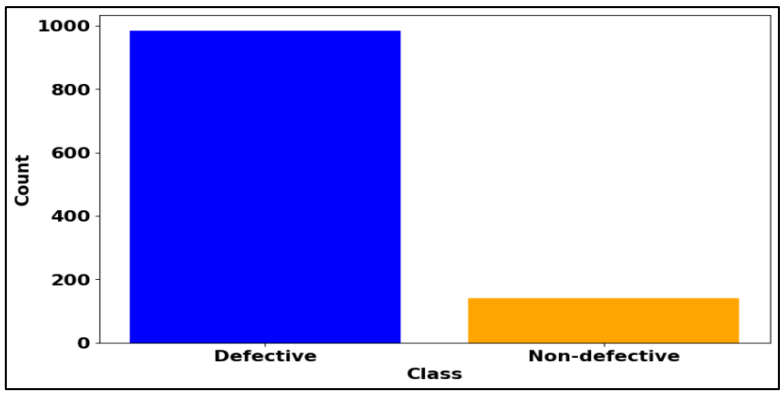

Figure 6. Class Distribution of Defective and Non-Defective Items

Figure 6 indicates that the dataset's faulty and non-defective items are distributed per class. The blue bar shows defective products, which are substantially higher in count, close to 1000, whereas the orange bar represents non-defective articles, with a much smaller count of, approximately 150–200. This suggests a class imbalance in which the dataset is dominated by defective objects. Model performance may be impacted by such an imbalance, necessitating the use of methods like weighted loss functions or resampling to increase classification accuracy.

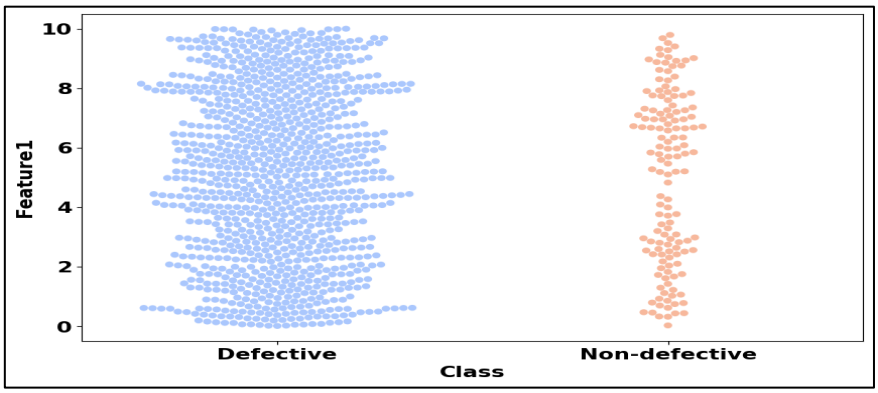

Figure 7. Feature Distribution Across Defective and Non-Defective Classes

Figure 7 displays the distribution of Feature 1 between the non-defective and defective groups. Defective products are represented by blue dots, which are widely distributed throughout the feature range and show considerable variance. On the other hand, non-defective items are represented by orange spots, which are more concentrated and indicate smaller variance. This suggests that the two classes behave differently in terms of features. By utilizing feature separability, these insights aid in feature selection and model training, enhancing classification performance.

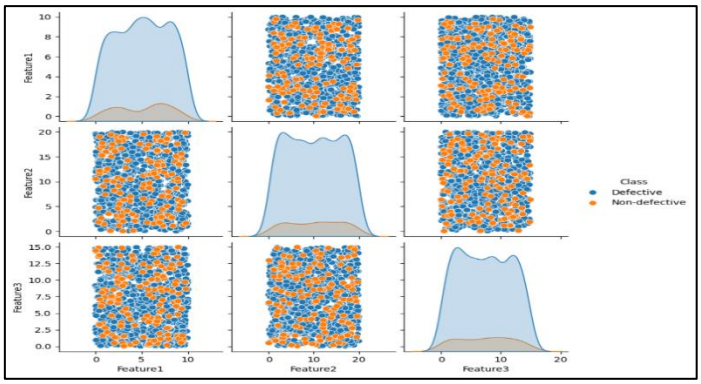

Figure 8. Pair Plot of Features for Defective and Non-Defective Class

Figure 8 illustrates a pair plot that indicates the relationships between several characteristics for both faulty and non-faulty items. Defective products are shown by blue points, and non-defective items are indicated by orange points. Each feature's distribution is displayed in diagonal plots, with defective items showing a wider spread than non-defective ones. In order to find patterns that can help with model training and classification tasks, this shows differences in feature distributions across the two classes.

Figure 9. Mean Value Comparison of Feature3 for Defective and Non-Defective Classes

A bar chart comparing the mean values of Feature3 for products that are defective and those that are not is depicted in the figure 8. Defective products are shown by the blue bar, and non-defective things are indicated by the orange bar. Feature 3 does not differentiate between defective and non-defective things, as evidenced by the nearly identical mean values for the two classes. This shows that Feature3 might have low predictive ability for categorization and may require additional analysis or feature engineering to improve its usefulness.

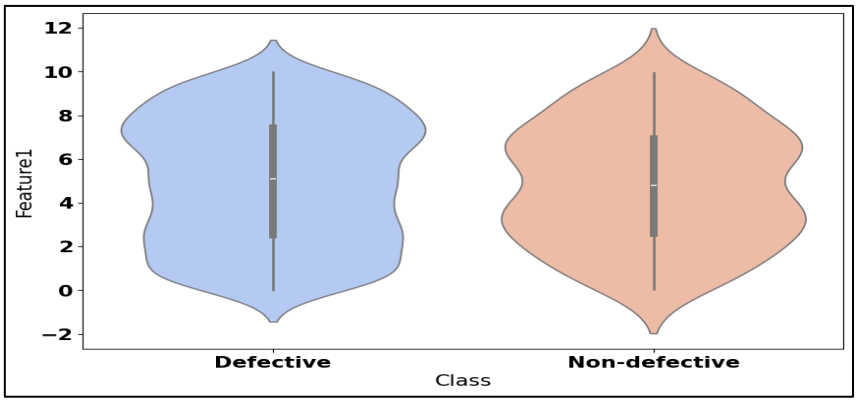

Figure 10. Violin Plot of Feature1 Distribution for Defective and Non-Defective Classes

A violin plot comparing the distribution of Feature1 for faulty and non-defective things is displayed in figure 10. Defective things are represented by the blue plot, while non-defective objects are represented by the orange plot. With a concentration of values in the middle range, both groups show a comparable spread. This suggests that Feature1 may not be a powerful differentiator for classification but still offers insight into variability, as it shows a similar distribution for both groups.

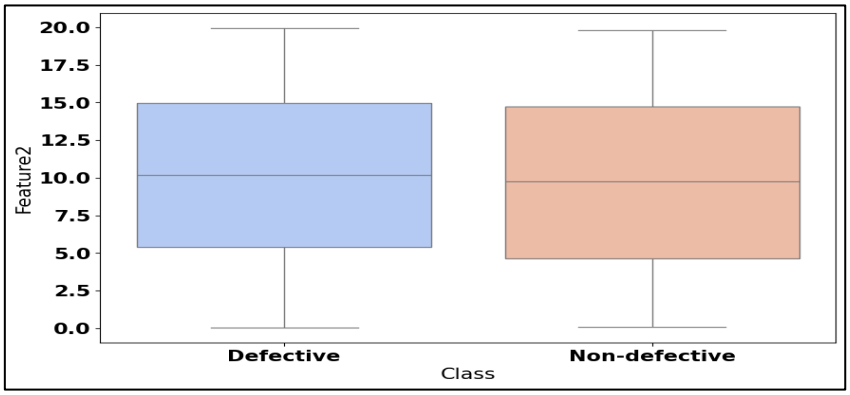

Figure 11. Box Plot of Feature2 for Defective and Non-Defective Classes

A box plot comparing the distribution of Feature2 for faulty and non-defective products is illustrated in figure 11. Defective things are indicated by the blue box, and non-defective items are indicated by the orange box. The median, interquartile range and overall spread of both classes are comparable, suggesting that Feature2 is distributed similarly in both categories. This suggests that while Feature2 might not be a powerful differentiator for categorization, further research might identify minute variations or interactions with other features.

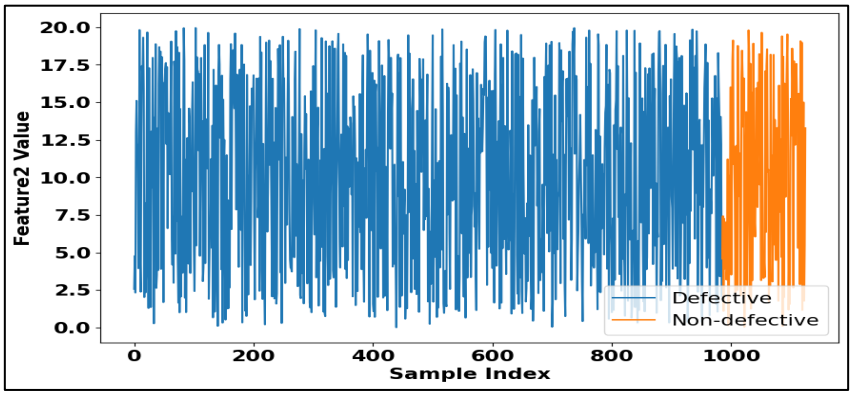

Figure 12. Feature2 Value Distribution Across Samples for Defective and Non-Defective Classes

A line plot of Feature 2 values for both defective and non-defective items across sample indices is shown in Figure 12. Defective products are indicated by blue lines, and non-defective things are indicated by orange lines. The majority of the collection contains defective items, and a minor portion contains non-defective items. This suggests a class imbalance, where defective samples predominate. Resampling strategies may be necessary to provide equitable categorization and enhance predictive accuracy, as well as to affect model performance.

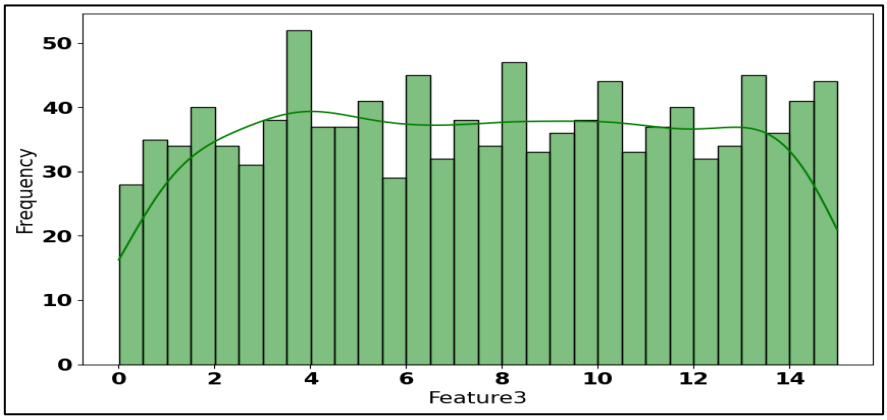

Figure 13. Distribution of Feature3 with Density Estimation

A histogram of Feature3 values, displaying their frequency distribution within the dataset, is depicted in the Figure 13. The kernel density estimate (KDE), which highlights the general distribution trend, is represented by the smooth green line, while the green bars show the number of observations falling within particular value ranges. A reasonably even range of Feature3 values is indicated by the histogram's apparent uniformity. In machine learning tasks, feature engineering, normalization, and model selection all depend on an understanding of such distributions.

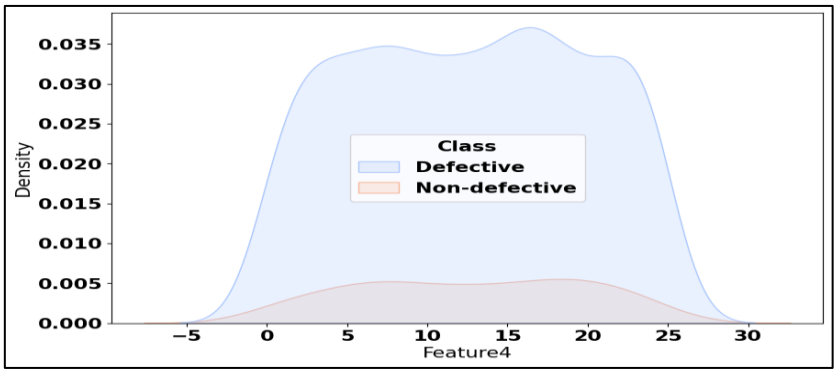

Figure 14. Kernel Density Estimation of Feature4 by Class

Figure 14 shows the probability density distribution of Feature 4 for both the defective and non-defective classes. Defective things are indicated by the blue shaded area, whereas non-defective items are shown by the light red area. Given its significantly higher density, the defective class appears to make up a bigger percentage of the sample. Classification performance may be impacted by the overlapping distribution, which indicates some resemblance between the two classes for this characteristic. Selecting features and preparing data for machine learning models is made easier with an understanding of feature distribution.

Figure 15. Correlation Matrix of Features

Figure 15 displays the correlation matrix of four features with values ranging from -1 to 1, each cell displays the Pearson correlation coefficient between two attributes. When the diagonal values are 1, complete self-correlation is indicated. Using the color gradient from blue (low correlation) to red (high correlation) makes relationships simpler to discern. All of the feature correlations in this case are near 0, indicating that there is either no linear association at all or very weak ones. This could have an effect on the choice of features in machine learning models.

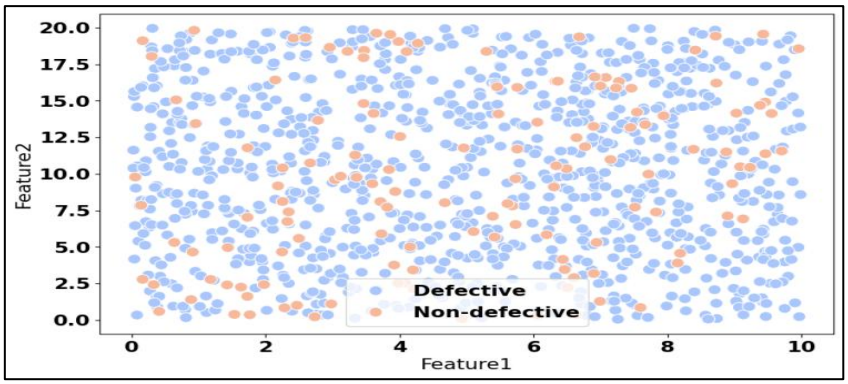

Figure 16. Scatter Plot of Feature1 vs. Feature2

The distribution of two characteristics, Feature1 and Feature 2, is shown in Figure 16. This scatter plot is categorized according to the class labels "Defective" (blue) and "Non-defective" (orange). There is no obvious pattern or linear division between the two classes, as the points seem to be dispersed randomly. This suggests that it could be difficult to differentiate between samples that are flawed and those that are not based solely on these characteristics. For improved class separation, more feature engineering or sophisticated classification techniques might be required.

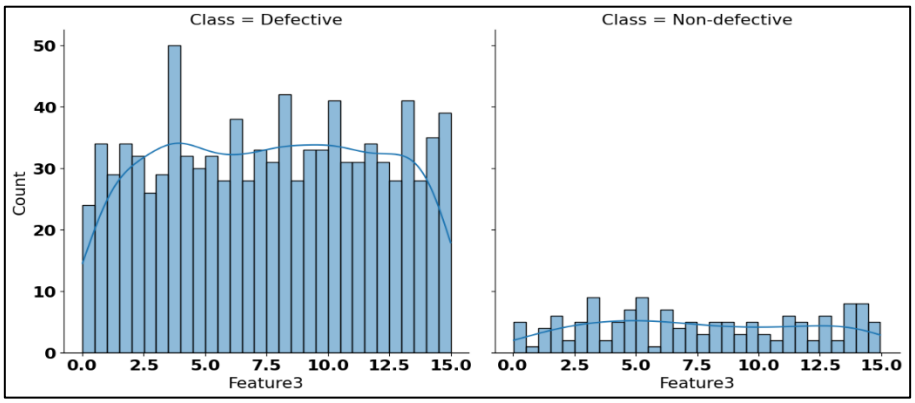

Figure 17. Distribution of Feature3 by Class

Figure 17 shows the distribution of Feature3 for the faulty and non-defective classes, Defective samples are depicted in the left plot, which has a somewhat uniform distribution with different peaks. Non-defective samples, which are substantially fewer and show a more distributed distribution, are depicted in the right plot. The two classes' glaringly different counts point to an imbalance in the data, which could affect classification performance. Strategies for feature selection and model improvement can benefit from an understanding of this distribution.

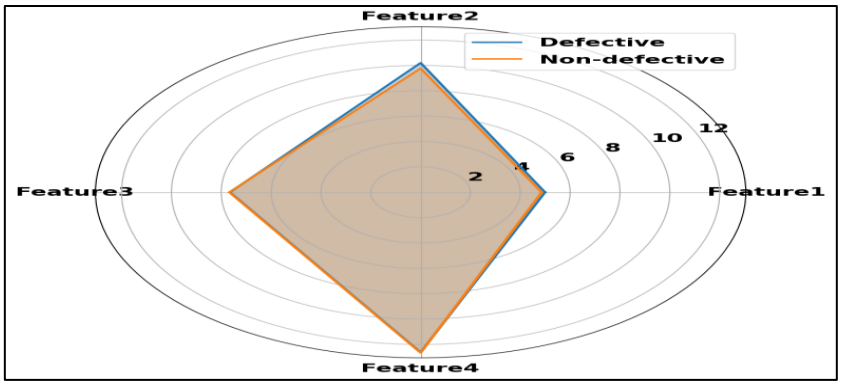

Figure 18. Radar Chart of Feature Comparison by Class

The average values of four characteristics (Feature 1, Feature 2, Feature 3, and Feature 4) for both defective and non-defective samples are provided in figure 18. Both classes have comparable distributions across these traits, as shown by the almost overlapping lines. Feature 1 displays the lowest values, while Features 2 and 4 show the highest. When examining multidimensional data, radar maps can be helpful in seeing trends and connections between several classes, which can aid in feature selection and the creation of classification models.

4.4 Performance Analysis of Proposed Method Compared with the Existing Methods

Using the PROMISE dataset, the efficacy and efficiency of the proposed approach are carefully assessed, and the outcomes are contrasted with those of other approaches. Precision, Accuracy, MCC, recall and F-measure are among the evaluation metrics. CNN [18], XGBoost [16], Bayesian Net [17], WACIL [19], and the Nested-stacking classifier [20] were the methods compared on the PROMISE dataset. These comparisons help assess how much the proposed method improves software defect prediction performance.

Table 6. Comparison of Proposed with Existing Methods for PROMISE Dataset

Methods	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**MCC (%)**	**F − Measure (%)**
XGBoost [16]	81.3	81.3	82.3	72.4	84.2
Bayesian Net [17]	77.6	78.5	90.4	72.1	83.8
CNN [18]	88.3	87.6	84.3	77.1	90.4
WACIL [19]	82.5	87.3	80.4	79.3	80.6
Nested-stacking classifier [20]	74.8	75.7	85.3	78.7	90.4
EQGAN-FbOA (Proposed)	99.8	99.7	99.6	99.5	99.4

A comparison of various software defect prediction techniques using the PROMISE dataset is shown in Table 6. Conventional methods with accuracy ranging from 74.8% to 88.3%, like CNN, Bayesian Net, and XGBoost, perform moderately. The proposed EQGAN-

FbOA model significantly outperforms all existing methods with impressive MCC of 99.5%, F-Measure of 99.4%, recall of 99.6%, precision of 99.7%, and accuracy of 99.8%. This demonstrates its exceptional efficacy in precisely forecasting software flaws with great performance and dependability.

4.5 Statistical Analysis of the Proposed Method Versus Existing Methods

The EQGAN-FbOA proposed approach is statistically evaluated against other baseline methods at a significance level of 0.05 for each project using Friedman's non-parametric test and Nemenyi's post-hoc test. By comparing the average ranks of the approaches, the Friedman test determines whether any differences are statistically significant. It does not presume any particular distribution and is less impacted by outliers because it ranks the approaches according to performance rather than raw values. This can be expressed in Equation (17).

$$\tau_{Y^2} = \frac{12M}{K(K+1)}\left(\sum_{i=1}^{K} CL_i^2 - \frac{K(K+1)^2}{4}\right) \tag{17}$$

where, K is the number of techniques that need to be compared in order for $CL_i^2 - \frac{K(K+1)^2}{4}$ to be calculated, and M is the total number of projects. L_i represents the rank of the $i-th$ project, and L_i represents the average rank of method i over all projects. Obedient to the Y^2 distribution, τ_{Y^2} has a degree of freedom of $K+1$. Then the variation of τ_F, is typically employed to perform the statistic test since the original Friedman test statistic is too conservative is shown in Equation (18).

$$\tau_F = \frac{(M-1)\tau_{Y^2}}{M(K-1)-\tau_{Y^2}} \tag{18}$$

where, the F-distribution is represented by the above equation, which has $(M-1)$ and the degrees of freedom, $(K-1)$.If τ_F is the lowest value from the F-distribution table, signifying statistically significant differences, the null hypothesis is rejected. A post-hoc test, like Nemenyi's test, can be used to determine whether specific procedures differ significantly if the null hypothesis is rejected. Equation (19) is used to determine the crucial difference DC

$$DC = h_{\alpha,K}\sqrt{\frac{K(K+1)}{6M}} \tag{19}$$

where, the number of techniques and significance level determine the crucial value, $h_{\alpha,K}$. Two approaches are deemed significantly different if their rank difference is more than the DC. The Nemenyi test's disadvantage is that approaches may fall under more than one overlapping category.

Table 7. Statistical Analysis Results Using Friedman Test and Nemenyi Post-hoc Test

Method	Average Rank	Rank Group
XGBoost [16]	4.33	Bottom
Bayesian Net [17]	5.17	Bottom
CNN [18]	3.00	Middle
WACIL [19]	3.50	Middle
Nested-stacking classifier [20]	2.17	Top
EQGAN-FbOA (Proposed)	1.83	Top

The statistical comparison of several approaches using the Nemenyi post-hoc test and the Friedman test is shown in Table 7. The average rank column shows the relative performance of each method across multiple datasets, with lower ranks indicating better performance. The proposed EQGAN-FbOA method achieved the best rank (1.83), followed by the Nested-Stacking classifier (2.17), both forming the top-performing group. CNN (3.00) and WACIL (3.50) fall into the middle group, while XGBoost (4.33) and Bayesian Net (5.17) perform the worst, forming the bottom group.

4.6 Ablation Study of the Proposed Method

Use the PROMISE dataset to assess the effects of each component in the proposed EQGAN-FbOA method. The baseline is the EQGAN model, highlighting its predictive function. When EQGAN is added, accuracy rises, proving how well attention processes work to improve predictions. The entire model, EQGAN-FbOA, achieves the highest accuracy,

highlighting the significance of FbOA-based weight parameter adjustment in improving software defect prediction performance.

Table 8. Ablation Study

Methods	PROMISE dataset	
	Accuracy (%)	**Precision (%)**
EQGAN	93.59	
EQGAN with SDT	92.97	
EQGAN - SDT- FbOA	99.9%	

An ablation study utilizing the PROMISE dataset to assess the effects of several EQGAN-FbOA model components is shown in Table 8. The EQGAN model alone achieves 93.59% accuracy, serving as the baseline. When incorporating SDT, the accuracy slightly decreases to 92.97%, indicating its limited contribution. The efficacy of FbOA-based weight parameter adjustment in improving software defect prediction performance is demonstrated by the entire model, EQGAN-SDT-FbOA, which performs noticeably better than the others with an astounding 99.9% accuracy.

4.7 Discussion

The PROMISE dataset was used to assess the EQGAN-FbOA model's capacity to forecast software faults. Performance analysis reveals that conventional methods, such as CNN, XGBoost, Bayesian Net, and WACIL, achieve moderate accuracy, with values ranging from 74.8% to 88.3%, In contrast, the proposed EQGAN-FbOA method achieves 99.8% accuracy, 99.7% precision, and 99.6% recall, demonstrating a significant improvement. Its superiority is confirmed by statistical validation using the Friedman test and Nemenyi post-hoc test, which position it as the best-performing model. Additionally, the ablation study highlights the contributions of EQGAN and FbOA, showing that FbOA-based weight optimization prevents premature convergence and enhances defect prediction accuracy. The study confirms that integrating quantum generative learning with football-inspired optimization leads to more reliable software quality assessment. These findings establish EQGAN-FbOA as a highly efficient approach for enhancing software defect prediction, improving accuracy, robustness, and overall software reliability.

5. Conclusion

The proposed EQGAN-FbOA framework significantly enhances software defect prediction by integrating Entangling Quantum Generative Adversarial Networks (EQGAN) with the Football Optimization Algorithm (FbOA). This model improves defect classification through quantum-enhanced feature representations, while FbOA optimizes feature selection and classification accuracy by effectively balancing exploration and exploitation. Experimental validation on the PROMISE dataset demonstrates that the proposed method outperforms conventional approaches, achieving accuracy of 99.8%, Matthews Correlation Coefficient (MCC) of 99.5%, precision of 99.7%, F-measure of 99.4%, and recall of 99.6%. However, the model presents certain limitations, including increased computational complexity with larger datasets, resulting in longer training times and resource constraints. Furthermore, the performance of FbOA is contingent upon hyperparameter tuning, necessitating additional optimization. Future research could investigate hybrid classical-quantum architectures to enhance computational efficiency and develop adaptive tuning mechanisms for FbOA. Extending this approach to multi-label defect classification may further broaden its applicability across various software engineering domains.

Reference

[1] Zheng, Wei, Tianren Shen, Xiang Chen, and Peiran Deng. "Interpretability application of the Just-in-Time software defect prediction model." Journal of Systems and Software 188 (2022): 111245.

[2] Uddin, Md Nasir, Bixin Li, Zafar Ali, Pavlos Kefalas, Inayat Khan, and Islam Zada. "Software defect prediction employing BiLSTM and BERT-based semantic feature." Soft Computing 26, no. 16 (2022): 7877-7891.

[3] Feng, Shuo, Jacky Keung, Yan Xiao, Peichang Zhang, Xiao Yu, and Xiaochun Cao. "Improving the undersampling technique by optimizing the termination condition for software defect prediction." Expert Systems with Applications 235 (2024): 121084.

[4] Pandit, Mahesha, Deepali Gupta, Divya Anand, Nitin Goyal, Hani Moaiteq Aljahdali, Arturo Ortega Mansilla, Seifedine Kadry, and Arun Kumar. "Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework." Applied Sciences 12, no. 1 (2022): 493.

[5] Azzeh, Mohammad, Yousef Elsheikh, Ali Bou Nassif, and Lefteris Angelis. "Examining the performance of kernel methods for software defect prediction based on support vector machine." Science of Computer Programming 226 (2023): 102916.

[6] Abdu, Ahmed, Zhengjun Zhai, Redhwan Algabri, Hakim A. Abdo, Kotiba Hamad, and Mugahed A. Al-antari. "Deep learning-based software defect prediction via semantic key features of source code—systematic survey." Mathematics 10, no. 17 (2022): 3120.

[7] Balasubramaniam, S., and Shantappa G. Gollagi. "Software defect prediction via optimal trained convolutional neural network." Advances in Engineering Software 169 (2022): 103138.

[8] Alazba, Amal, and Hamoud Aljamaan. "Software defect prediction using stacking generalization of optimized tree-based ensembles." Applied Sciences 12, no. 9 (2022): 4577.

[9] Alazba, Amal, and Hamoud Aljamaan. "Software defect prediction using stacking generalization of optimized tree-based ensembles." Applied Sciences 12, no. 9 (2022): 4577.

[10] Liu, Jingyu, Jun Ai, Minyan Lu, Jie Wang, and Haoxiang Shi. "Semantic feature learning for software defect prediction from source code and external knowledge." Journal of Systems and Software 204 (2023): 111753.

[11] Bai, Jiaojiao, Jingdong Jia, and Luiz Fernando Capretz. "A three-stage transfer learning framework for multi-source cross-project software defect prediction." Information and Software Technology 150 (2022): 106985.

[12] Zivkovic, Tamara, Bosko Nikolic, Vladimir Simic, Dragan Pamucar, and Nebojsa Bacanin. "Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on shapley additive explanations." Applied Soft Computing 146 (2023): 110659.

[13] Nevendra, Meetesh, and Pradeep Singh. "Empirical investigation of hyperparameter optimization for software defect count prediction." Expert Systems with Applications 191 (2022): 116217.

[14] Tang, Yu, Qi Dai, Mengyuan Yang, Tony Du, and Lifang Chen. "Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm." International Journal of Machine Learning and Cybernetics 14, no. 6 (2023): 1967-1987.

[15] Bai, Jiaojiao, Jingdong Jia, and Luiz Fernando Capretz. "A three-stage transfer learning framework for multi-source cross-project software defect prediction." Information and Software Technology 150 (2022): 106985.

[16] Dar, Abdul Waheed, and Sheikh Umar Farooq. "An ensemble model for addressing class imbalance and class overlap in software defect prediction." International Journal of System Assurance Engineering and Management 15, no. 12 (2024): 5584-5603.

[17] Mehmood, Iqra, Sidra Shahid, Hameed Hussain, Inayat Khan, Shafiq Ahmad, Shahid Rahman, Najeeb Ullah, and Shamsul Huda. "A novel approach to improve software defect prediction accuracy using machine learning." IEEE Access 11 (2023): 63579-63597.

[18] Khleel, Nasraldeen Alnor Adam, and Károly Nehéz. "A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method." Journal of Intelligent Information Systems 60, no. 3 (2023): 673-707

[19] Manchala, Pravali, and Manjubala Bisi. "Diversity based imbalance learning approach for software fault prediction using machine learning models." Applied Soft Computing 124 (2022): 109069.

[20] Chen, Li-qiong, Can Wang, and Shi-long Song. "Software defect prediction based on nested-stacking and heterogeneous feature selection." Complex & Intelligent Systems 8, no. 4 (2022): 3333-3348.

[21] Mehmood, Iqra, Sidra Shahid, Hameed Hussain, Inayat Khan, Shafiq Ahmad, Shahid Rahman, Najeeb Ullah, and Shamsul Huda. "A novel approach to improve software defect prediction accuracy using machine learning." IEEE Access 11 (2023): 63579-63597.

[22] Ali, Misbah, Tehseen Mazhar, Yasir Arif, Shaha Al-Otaibi, Yazeed Yasin Ghadi, Tariq Shahzad, Muhammad Amir Khan, and Habib Hamam. "Software defect prediction using an intelligent ensemble-based model." IEEe Access 12 (2024): 20376-20395.

[23] Aftab, Shabib, Sagheer Abbas, Taher M. Ghazal, Munir Ahmad, Hussam Al Hamadi, Chan Yeob Yeun, and Muhammad Adnan Khan. "A cloud-based software defect prediction system using data and decision-level machine learning fusion." Mathematics 11, no. 3 (2023): 632.

[24] Alkhasawneh, Mutasem Shabeb. "Software defect prediction through neural network and feature selections." Applied Computational Intelligence and Soft Computing 2022, no. 1 (2022): 2581832.

[25] Thirumoorthy, K., and J. Jerold John Britto. "A clustering approach for software defect prediction using hybrid social mimic optimization algorithm." Computing 104, no. 12 (2022): 2605-2633.

[26] Zheng, Shuhan, and Nontawat Charoenphakdee. "Diffusion models for missing value imputation in tabular data." arXiv preprint arXiv:2210.17128 (2022).

[27] Yao, Man, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. "Spike-driven transformer." Advances in neural information processing systems 36 (2023): 64043-64058.

[28] Niu, Murphy Yuezhen, Alexander Zlokapa, Michael Broughton, Sergio Boixo, Masoud Mohseni, Vadim Smelyanskyi, and Hartmut Neven. "Entangling quantum generative adversarial networks." Physical Review Letters 128, no. 22 (2022): 220505.

[29] El-Kenawy, El-Sayed M., Faris H. Rizk, Ahmed Mohamed Zaki, M. E. Mohamed, A. Ibrahim, Abdelaziz A. Abdelhamid, Nima Khodadadi, E. M. Almetwally, and M. M. Eid. "Football optimization algorithm (fboa): A novel metaheuristic inspired by team strategy dynamics." J. Artif. Intell. Metaheuristics 8, no. 1 (2024): 21-38.