Software Failure Prediction Based on Game Theory and Convolutional Neural Network Optimized by Cat Hunting Optimization (CHO) Algorithm

Azam Ghaedi[1], Amid Khatibi Bardsiri[2]* , Mehdi Jafari Shahbazzadeh[3]

1.PHD Student, Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran.
2.Assistant Professor, Department of Computer Engineering, Bardsir Branch, Islamic Azad University, Kerman, Iran (Corresponding author).
3.Assistant Professor, Department of Electrical Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran.

*** Corresponding author email address**: a.khatibi@srbiau.ac.ir

Received: 2024-12-03 **Reviewed:** 2025-01-01 **Revised:** 2025-01-11 **Accepted:** 2025-01-20 **Published:** 2025-03-29

Abstract

Predicting the failure of software projects in the stages of software production reduces the losses of software companies. Deep learning methods are an efficient tool for software failure prediction. Imbalance of data sets, intelligent feature selection, and accurate deep learning techniques are among the challenges of deep learning methods for accurate software failure prediction. This manuscript presents an improved game theory method based on the Generative adversarial (GAN) network to predict software failure and success. In the second stage, the cat-hunting algorithm is used to select features and reduce the dimensions of the samples. The dimensionally reduced samples are converted into RGB color images in the third step. RGB images are used for convolutional neural network(CNN) training. The advantage of the proposed method is to intelligently select the feature, reduce the input of the CNN neural network, and simultaneously balance the training samples with game theory to increase the accuracy of the prediction model. In this manuscript, the NASA dataset is used to predict the failure of software projects. The accuracy of the proposed method (GCV) in predicting the failure of software projects is equal to 96.69%. The GCV method is more accurate in predicting the failure of software projects than the LSTM and VGG16 methods. The proposed method is more accurate in feature selection than Chi, IG, and ReF WOA, HHO, and JSO algorithms methods.

Keywords: *Software failure prediction, Feature selection, Cat Hunting algorithm, GAN, SMOTE, CNN.*

How to cite this article:

Ghaedi A, Khatibi Bardsiri A, M, Jafari Shahbazzadeh M. (2025). Software Failure Prediction Based on Game Theory and Convolutional Neural Network Optimized by Cat Hunting Optimization (CHO) Algorithm. Management Strategies and Engineering Sciences, 7(1), 34-55.

1. Introduction

Many software projects are offered in the very complex and modern world today. Complex and modernized software is done during the engineering process of software production. It takes a lot of money to produce complex software. Big software projects are developed by big companies [1]. Many software projects succeed by spending much money, and some projects have defects [2]. Failure to achieve predetermined software production goals is considered software project failure [3]. In some software projects, project failure costs millions of dollars [4].

With the complexity of software design, ensuring the software's reliability is very important [5]. A lot of testing and debugging is running to design a reliable software engineering project. One of the challenges of traditional approaches to predicting the failure of software projects is the time-consuming process [6]. Another challenge of these methods is that the software must be completed and tested. Creating a software project that ultimately leads to failure can cost software production companies much [7]. A suitable approach is to predict the success or failure of the project at the time of software production [8]. Predicting the failure of a software project in the early stages will prevent software development companies from spending money, and time will be well spent. Another advantage of predicting the success or failure of software projects at each stage of project development is that the factors leading to failure can be recognized [9].

Software defect prediction is a technique to improve software quality and reduce software testing costs with predictive methods such as machine learning [10] and deep learning [11]. Many software companies use prediction methods to predict success in customer satisfaction and save software production costs [12]. Software defect prediction is a vital component of the life cycle process of application software development [13]. Anticipating software defects makes it possible to quickly provide customers with high-quality and low-cost software. At each stage of software production, a quality software product can be provided by predicting the success of the goals. Many companies producing software are inclined to predict software defects at the very early stages of software development to maintain the quality of the software for customer satisfaction and save the cost of testing [14].

Software effort estimation is considered a primary activity in software project management. Software failure prediction is one of the tools for estimating effort in software development [15]. Studies show that many software defects are due to inappropriate project management methods. Various factors contribute to the failure of a software project, including unrealistic and unexpected goals for the project, incorrect estimation of required resources, incorrect definition of requirements, poor reporting of project development, and poor communication between the programming team. , poor communication between developers and users, poor technology, inability to manage the programming team, use of non-standard development methods, little experience of the programming team, and lack of cost measurement [1, 4].

Project effort estimation includes estimating software development's size, effort, cost, time, and staffing. The product's success or failure is often estimated at the initial stage of any software development project. Predicting a software project's failure or success is considered an activity in estimating the software effort to complete a project [16]. Estimates show that many software projects fail, and this issue imposes a significant loss on software developers [17]. Predicting the failure of a software project is essential and necessary because it produces reliable and quality software [18]. Predicting the defeat or success of a software project with prediction methods such as machine learning and deep learning, unlike traditional approaches, requires less time and cost [18].

Machine learning and deep learning methods play an essential role in predicting the failure of software projects. Learning methods create a classification or prediction model through the training process, which is used to predict the success or failure of software projects. Machine learning methods such as SVM [19], neural network [20], decision tree [21], random forest [22], and regression [23] have been used to predict the failure of software projects. Deep learning methods provide a higher level of machine learning and are widely used to create models for predicting the failure and success of software projects [24]. Convolutional neural networks [25], Long short-term memory (LSTM)[26], and transformers [27] are deep learning models for predicting the failure of software projects.

One of the fundamental challenges of predicting the failure of software projects is the small number of samples in the data set. The small number of samples used to train machine learning and deep learning models leads to the production of weak prediction models [28]. Another fundamental challenge in developing software project failure and success prediction models is the need for feature selection. Most studies use all features to create a software failure prediction model. The feature selection makes learning the model on the primary and optimal features and reduces the prediction error [29]. Deep learning methods such as CNN are mainly used for image processing and are less used for predicting software success and failure. To solve this challenge, unique and new coding is needed on the data of software projects to use the CNN method to predict the failure and success of the software.

This manuscript presents an advanced method combining game theory, swarm intelligence, and deep learning to predict the failure and success of software projects. In the first stage, the proposed method combines the GAN deep learning method [30] and the SMOTE method [31] to increase the number of data set samples and balance it. GAN neural network is a suitable method for generating artificial samples based on actual and game theory samples. The Cat Hunting Optimization (CHO) algorithm [32] developed by the authors is used for feature selection. The selected feature vector map on the dataset is in the third step. The given mapping data is then coded into RGB color images and placed as the input of the CNN neural network and VGG 16 architecture. The output of the VGG 16 deep learning model has two states: failure and success.

The main goal of the manuscript is to estimate and predict the failure of software projects with deep learning and swarm intelligence. The proposed method aims to discover failures in the early stages of software production, reduce the losses of software developers, and increase customer satisfaction. The contributions of the authors in the manuscript and the proposed method are as follows:

- Improving the GAN deep learning technique with the SMOTE method in balancing data sets related to software projects.
- Creating artificial examples of software projects with game theory
- Improving the cat optimization algorithm for feature selection in detecting the basic features of software projects
- Using a new coding to train the VGG 16 neural network in predicting the failure of software projects

The manuscript is presented in five sections. Section II reviews the subject literature on predicting the failure of software projects and related works. Section III presents the failure prediction model of software projects by integrating deep learning and a streamlined algorithm. In section IV, the failure prediction model of software projects is implemented and analyzed. In section V, the results are discussed, and future works are reviewed.

2. Related works

According to reports published in 2018, more than 50% of the total cost of software production is used to identify and fix defects and losses caused by software project defects [7]. However, with the potential of machine learning to predict software failures, these costs could be significantly reduced. Reports show that 1.7 trillion dollars were spent to fix these violations, and 314 software companies have been involved [7]. If the software is used in critical systems, its violation can be destructive. For example, the role of software in an airplane can cause a disaster [33]. Software violations in nuclear power control systems and industries cause significant losses and disasters [34, 35]. Deep learning and code analysis methods of software projects are powerful tools for predicting software failure [36]. LSTM, DBN, CNN, GNN, and BERT are deep learning methods applied to software defect prediction. This section reviews related works in software defect prediction with machine learning (ML) and deep learning (DL). In the rest of this section, several studies on software failure with deep learning and machine learning approaches are reviewed.

In [19], Software defect prediction is presented based on support vector machines with different kernels. Experiments show that using 40% of features with all kernel functions works best. In [1], software defect prediction methods are investigated using combined techniques. Their main advantage is that they review many papers. A challenge of their method is the need for more consideration of data set balancing and feature selection methods. In [37], they provide a software defect prediction method based on the Variable sparrow search algorithm. One challenge of their proposed method is the imbalance of the data set. In [14], optimal machine learning techniques for software error prediction are suggested. Their evaluation shows that their method is more accurate than SVM, Naive Bayes, and the nearest neighbor in predicting software defects.

In [38], a feature transfer learning method with reinforcement learning is present for software defect detection. The proposed SDP method uses feature transfer learning to map original features to another feature space. They evaluated the proposed method on 43 projects from the PROMISE and NASA MDP datasets using three classifiers: logistic regression, random forest, and Bayesian network. Experimental results show that their method is more accurate than logistic regression, random forest, and Bayesian networks. In [39], they presented a software defect prediction method using a tree-based method. This research uses grid search to optimize meta-parameters of random forest methods, Extra trees, AdaBoost, gradient boosting, histogram-based gradient boosting, XGBoost, and CatBoost. Their method uses 21 software defect datasets to evaluate. The experimental results showed that Extra trees and random forests have the highest accuracy in predicting software violations.

In [40], they presented a software defect prediction model based on a complex and graph neural network. This research proposes a software defect prediction framework based on graph neural networks and complex networks:

1. They code the software as a graph where nodes represent classes and edges represent dependencies between classes.
2. The community detection algorithm divides the graph into several subgraphs.
3. The improved graph neural network model learns the vector representing nodes.

The proposed model is implemented on the PROMISE dataset. Their method is more accurate than a graph-based neural network. The challenge of their method is the high complexity of the model. In [41], a software defect prediction method based on deep learning is proposed for mobile applications. They used deep learning algorithms like CNN and LSTM to develop a defect prediction model for Android-based applications. Based on the results, the CNN-based model has the best performance for predicting mobile phone app defects, and its accuracy is around 93.3%. In [42], software defect prediction using artificial neural networks is reviewed. This study's advantage is that it reviews articles in the field of neural networks, and its challenge is not to review other deep learning methods.

In [43], software defect prediction presents a balancing method based on KNN. Experiments use benchmark data sets from the NASA repository, including CM1, JM1, KC1, KC2, and PC1. The evaluation classifier showed that the proposed model has an accuracy of 96.9% and a confidence level of 95%. In [44], a software defect prediction approach using support vector machines is presented. The proposed method improves accuracy by 16.73% compared to the support vector machine. In [45], a genetic algorithm-based sampling method for balancing classes in software defect prediction is presented. This method is compared to several existing algorithms such as SMOTE, BSMOTE, ADASYN, over-random sampling, and MAHAKIL balancing algorithms. The results show that their proposed algorithm outperforms these methods regarding prediction error reduction.

In [46], software defect prediction is based on an advanced extreme learning machine present. The proposed method is based on the SSDAE and extreme learning optimized by particle swarm optimization (PSO) and gravitational search algorithm (GSA) in this research. In [47], they presented a software defect prediction method using the Island Moth Flame Optimization. The experiments' results show that feature selection using the IsBMFO algorithm improves the classification results and provides the best results in combination with the support vector machine . In [48], they presented a software defect prediction method based on a combined particle swarm optimization and sparrow search algorithm. This research combines PSO and SSA to improve convergence. The experimental results showed that SSA-PSO has less error than SSA and PSO algorithms in predicting software defects. In [49], a new multi-objective optimization algorithm is presented to predict software defects. In this model, defect detection and false alarm rates are considered two goals in software failure prediction. In [50], random forest algorithms and multi-objective optimization are proposed for predicting software defects. The test showed that the proposed method improved the AUC index by 2.78 and 3.46 compared to MONB and MONBNN, respectively.

In [51], they present an optimized machine-learning model for predicting software bugs. The principal component analysis (PCA) method reduces dimensions and selects features. The tests showed that their method of predicting software bug detection has an accuracy of about 97.8%. In [52], a genetic algorithm-based feature selection method for software defect prediction using SVM is presented. This research proposes a genetic evolution (GeEv) technique for feature selection. The experimental results show that the GeEv method performs better than the traditional genetic algorithm approach and can provide better statistical accuracy in software prediction. In [53], a software defect prediction method based on genetic evolution based on the 3-parent child is presented. For software defect prediction (SDP), data with large dimensions are used, so selecting features to reduce the dimensions is recommended. Experiments showed that their proposed method of filter-based feature selection techniques and wrapper-based feature selection techniques up to 17.5% AUC index. In [51], they presented a feature selection method based on the Firefly algorithm for software error prediction. This study uses the NASA collection that is available to the public. Evaluations showed that their method is more accurate in predicting software failure than methods such as GA and PSO algorithms. Table (1) reviews a summary of related works with their advantages and disadvantages.

Table 1. Advantages and disadvantages of related works

Source	Method	Advantage	defect
[19]	Support vector machine with different kernels	60% reduction in dimensions	Failure to check more complex kernels
[1]	Software defect prediction using hybrid techniques	Review a large number of methods	Failure to review balancing methods and feature selection
[37]	Prediction of software defects based on Sparrow's adaptive variable search algorithm	Prediction accuracy in 15 data sets is higher than in similar methods.	Unbalanced data set
[14]	Optimal machine learning techniques for software error prediction	More accuracy than SVM, Naive Bayes, and KNN methods.	Unbalanced data set
[38]	A feature transfer learning method with reinforcement learning	More accurate than logistic regression, random forest, and Bayesian network	medium accuracy
[39]	Software defect prediction using tree methods	Optimization of machine learning parameters	No balancing and no feature selection
[40]	Complex network and graph neural network	More accuracy than graph-based neural network	High complexity of the model
[41]	CNN and LSTM	The prediction accuracy is about 93.3%	Time consuming training
[42]	Review of methods based on artificial neural networks	A comprehensive review of papers in the field of neural networks	Failure to investigate other deep learning methods
[43]	Balancing based on KNN	High accuracy of about 96.9%	No reduction in machine learning input
[44]	Filtered support vector machines	The prediction accuracy compared to the SVM improved by 16.73%	No balancing and no feature selection
[45]	Genetic algorithm for balancing	Performance better than SMOTE, BSMOTE, ADASYN	Uncertainty of meta-heuristic algorithms
[46]	Extreme learning	Discover deep semantic features	High model complexity
[47]	Island moth flame optimization algorithm+SVM	More accuracy than SVM method	Unbalanced data set
[48]	Software defect prediction based on PSO and SSA algorithms	Less prediction error than SSA and PSO algorithm	More prediction time than SSA and PSO algorithm
[49]	Software defect prediction by multi-objective optimization algorithm	Reducing prediction error	Uncertainty
[50]	Random forest algorithms and multi-objective optimization for software defect prediction	More accuracy than MONB and MONBNN methods	Lack of feature selection phase
[51]	Feature selection by PCA method	High accuracy	Unbalanced data set
[52]	Feature selection by three-parent genetic algorithm	More accuracy than genetic algorithm	Unbalanced data set
[53]	NSGA-II methods for software defect prediction	More accurate than methods based on filter and wrapper	Unbalanced data set
[51]	Feature selection by Firefly algorithm	More accuracy than GA,DE, and PSO algorithm	Unbalanced data set

The review of related works shows that most studies need a mechanism for balancing the data set to accurately predict software failure. Meta-heuristic algorithms such as PSO and GA are used in the studies, but these algorithms need to better model global and local search balances. In the research, advanced convolutional neural network architectures have yet to be used to accurately predict software defect. The proposed method reduces the software defect prediction error by balancing the data set with game theory and the deep learning method based on a convolutional neural network. In the proposed method to increase the prediction model's performance, the swarm intelligence of cats is also used in hunting so that learning is done on more optimal features and the error of the software failure prediction model is reduced.

3. The proposed method

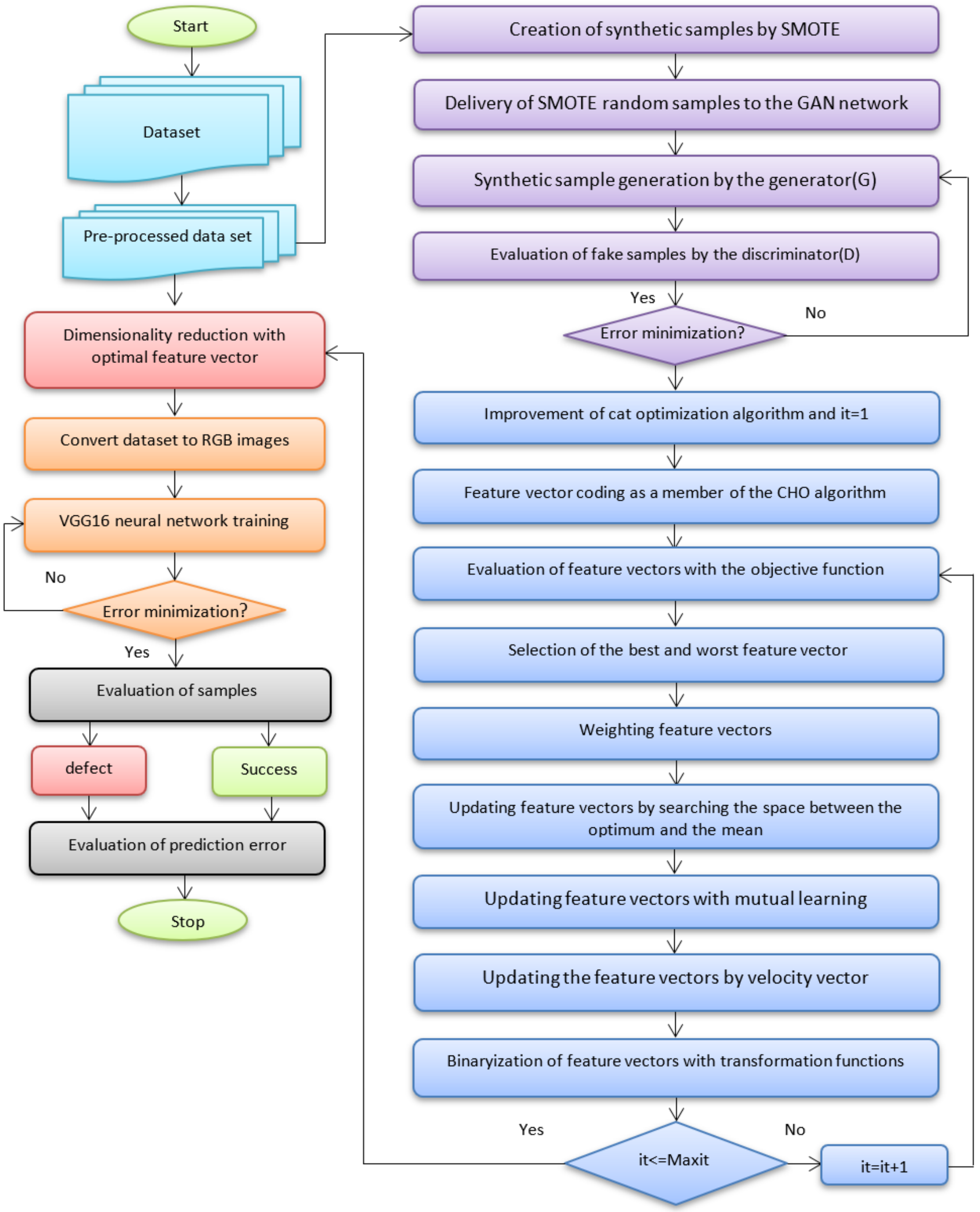
This section presents a proposed method or GAN+CHO+VGG16 (GCV) for software defect prediction with an improved cat optimization algorithm and deep learning. The innovation of the proposed method is as follows:

- The SMOTE algorithm is used to improve GAN deep learning in the first step.

- Hybrid the SMOTE algorithm and deep learning based on game theory with the GAN network balances the dataset.
- The cat optimization algorithm is improved, the hunting mechanism is added, and a binary version is provided for feature selection.
- The data selected in the dataset are converted into RGB color images by the feature vector and used to train the VGG16 neural network.

3.1. *The proposed framework*

The proposed framework for software failure prediction is shown in Figure 1. According to the proposed framework for software failure prediction, the following steps are presented:



Start
Dataset
Pre-processed data set
Dimensionality reduction with optimal feature vector
Convert dataset to RGB images
VGG16 neural network training
No
Error minimization?
Yes
Evaluation of samples
defect
Success
Evaluation of prediction error
Stop
Creation of synthetic samples by SMOTE
Delivery of SMOTE random samples to the GAN network
Synthetic sample generation by the generator(G)
Evaluation of fake samples by the discriminator(D)
Yes
No
Error minimization?
Improvement of cat optimization algorithm and it=1
Feature vector coding as a member of the CHO algorithm
Evaluation of feature vectors with the objective function
Selection of the best and worst feature vector
Weighting feature vectors
Updating feature vectors by searching the space between the optimum and the mean
Updating feature vectors with mutual learning
Updating the feature vectors by velocity vector
Binaryization of feature vectors with transformation functions
Yes
No
it<=Maxit
it=it+1

Figure 1. The framework of the proposed method for software defect prediction

- Data set balancing with GAN and SMOTE neural network.
- Selection of essential features with improved cat optimization algorithm.
- Reducing the dimensions of data set samples.
- Converting numerical samples to RGB images.
- VGG16 neural network training.
- Classification of software projects into two classes of failure and success with the trained network VGG16

3.2. *Preprocessing dataset*

The dataset used for software defect prediction has a large number of features. Each of the features has a range, and these ranges are different from each other. The normalization process is used in the pre-processing phase so that the upper and lower limits of the values of all features are the same and normalized. Equation (1) is used for pre-processing and normalization in the [a,b] range:

1

$$x_{normal} = a + (b - a)\frac{x - \min(x)}{\max(x) - \min(x)}$$

In this equation, x is the value of an unnormalized feature, and x_{normal} is the normalized value of feature x.

3.3. *Balancing the data set*

GAN network is a deep learning technique based on game theory. The generator(G) and discriminator (D) are the two main parts of this neural network. There is a game between the generator and the discriminator, and its goal is that the generator succeeds in deceiving the discriminator. Based on the real samples, the manufacturer tries to produce artificial or fake samples and deliver them to a discriminator. The generator wins if the discriminator is deceived and puts the artificial and fake samples in the real class. If the discriminator succeeds in placing the fake sample in the fake category, then the discriminator wins [54]. In [9], the SMOTE algorithm is used to improve and increase the accuracy of the GAN network. The role of SMOTE is to generate random and artificial samples from the minority class for the generator in the GAN network. The SMOTE algorithm increases the diversity of generated samples and the quality of the GAN network when producing synthetic samples. In Figure 2, SMOTE-GAN is shown for generating synthetic samples and increasing the number of software project datasets' samples. The SMOTE method for producing synthetic samples has challenges. The main disadvantage of SMOTE is that it focuses too much on local information and neighborhood data, so it does not produce a diverse set of new data [55].

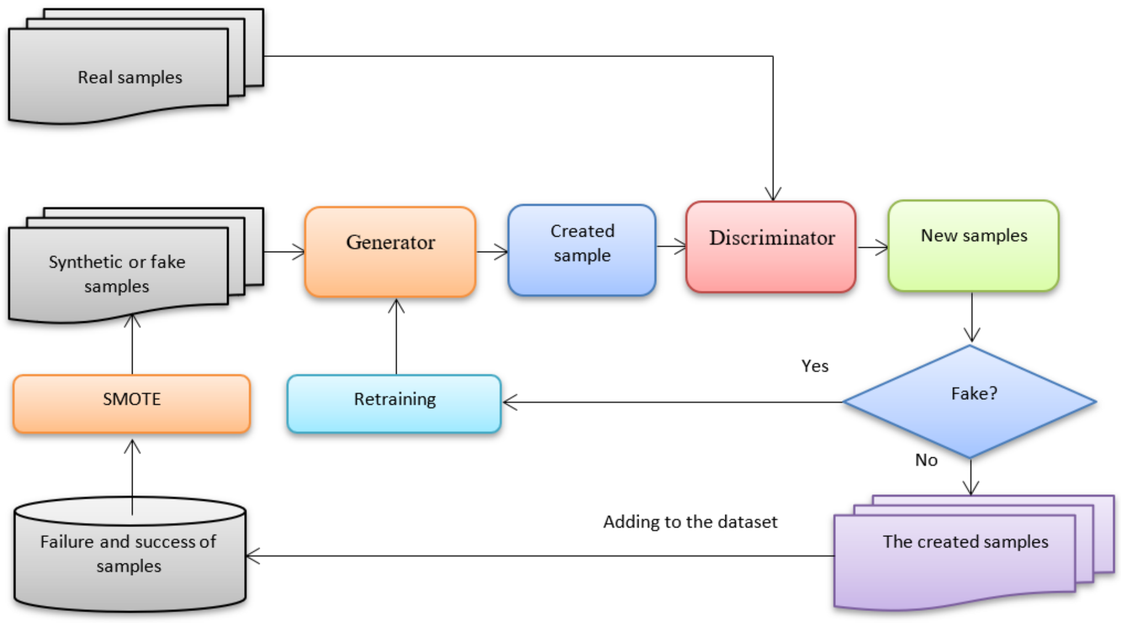


Figure 2. Generation of random samples by GAN + SMOTE

In the SMOTE method, several Real samples are blindly selected, and several random samples are created by interpolation. The SMOTE method produces artificial samples with little diversity and makes the generated data contribute little to creating accurate classification and prediction models. Unlike SMOTE, the GAN network produces more random data. By combining SMOTE and GAN methods, the created samples become more realistic because GAN creates global data to a large extent, and SMOTE creates local data. In the improved GAN model, the random and Real input of the generator is considered with x and z, respectively. The objective function in the discriminator is presented in the form of Equation (2) [9]:

2
$$\max_D \mathbb{E}_x[\log D(x)] + \mathbb{E}_z[\log(1 - D(G(z)))]$$

The objective function of the discriminator in Equation (2) is of the maximization type. Equation (3) to minimize the objective function of the discriminator is suggested [9]:

3
$$\min_D \mathbb{E}_x[-\log D(x)] - \mathbb{E}_z[\log(1 - D(G(z)))]$$

D(x) contains the output probabilities of the discriminator for the Real data, and D(G(z,x)) contains the output probabilities of the discriminator for the generated data. The objective function in the generator is defined as Equation (4) [9]:

4
$$\min_G - \mathbb{E}_z[\log D(G(z))]$$

To increase the efficiency of the proposed model in the discriminator and generating part, sigmoid activity functions according to Equations (5) and (6) are used [9]:

5
$$\mathbb{E}_x[-\log(1 + e^{-y})] - \mathbb{E}_z[1 - \log(1 + e^{-\hat{y}})]$$
6
$$\mathbb{E}_z[-\log(1 + e^{-\hat{y}})]$$

y and $\hat{y}$ are the outputs of the discriminator and generator, respectively, before applying the activation function. In the improved version, the GAN random generator function is replaced with minority samples produced by SMOTE, and the objective function in the discriminator and generator part is formulated as Equations (7) and (8):

7
$$\max_D \mathbb{E}_{x^*}[\log D(x^* \mid x)] + \mathbb{E}_u[\log(1 - D(G(u)))]$$
8
$$\min_G - \mathbb{E}_u[\log D(G(u))]$$

In these equations, x^* is the training samples of the minority class, and u is the oversampling data of the same class generated from various algorithms, such as SMOTE.

3.4. *Feature selection Cat Hunting Optimization (CHO) algorithm*

The cat optimization algorithm is a swarm intelligence algorithm. The cat optimization algorithm contains the concepts of the particle swarm optimization algorithm. In the optimization algorithm, every problem solution is considered a cat. In the standard version of the cat optimization algorithm, the distinction between solutions based on merit is not considered. In the proposed method, each cat or problem solver with more competence and experience can search around his space to find the optimal solution. The Cat Hunting Optimization(CHO) algorithm[32] presented by the authors in 2023, unlike the cat algorithm, has more intelligence, and Experiments show that this algorithm has less error in finding the optimal solution of optimization problems than PSO, DE, FA, GOA, SHO, MFO, and WOA algorithms. The advantage of the CHO algorithm is as follows, and for this reason, a binary version of it is presented in this manuscript for feature selection[32]:

- The ability to explore, search, and perform dynamic exploitation.
- The ability to search for the space between the optimum and the population mean.
- Giving weight to more appropriate solutions for further searching around these solutions.
- Using trigonometric equations for more adequate formulation.
- Learning between solutions.

Each problem solution is a feature vector in the proposed method, and a population of random feature vectors is created according to Equation (9).

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & x_{1,\text{d-1}} & x_{1,\text{d}} \\ x_{2,1} & \cdots & x_{2,j} & \cdots & x_{2,\text{d}} \\ \cdots & \cdots & x_{i,j} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{N-1,1} & \cdots & x_{N-1,j} & \cdots & x_{N-1,\text{d}} \\ x_{N,1} & \cdots & x_{N,j} & x_{N,\text{d-1}} & x_{N,d} \end{bmatrix} \quad \textbf{9}$$

In this equation, d is the number of dimensions of the problem and the number of elements of each feature vector. N is the number of feature vectors for software defect prediction. Each matrix row is a feature vector with zero and one element, and the matrix's columns are the dataset's features.

The feature number j of the feature vector number X_{ij} represents me. Each feature vector needs an appropriate objective function to evaluate. This research considers two error elements and the number of features to evaluate the feature vectors. The proposed objective function is a linear function of the error component and the number of features according to Equation (10).

$$Cost(X_i) = w_1 \times \frac{1}{n} E(X_i) + w_2 \times \frac{\|X_i\|}{d} \quad 10$$

X_i is the number of features selected by X_i, and E(X_i) is the software defect prediction error by X_i. Each feature vector in this step is set as the input of the MLP neural network, and its output error is used for evaluation. w_1 and w_2 are two random weight coefficients in the range [0,1]. In the proposed method, the most optimal and the worst population cats consider b and w, respectively, assuming it is a maximization problem. A cat with f(b) competence is the most experienced cat, and a cat with f(w) competence has the worst competence and has little experience. Any cat with more experience can consider more states for tracking. If a cat has little experience and competence, it considers fewer states for tracking and chooses one.

Suppose that cat or solution b and w have the number of states $State_{max}$ and $State_{min}$ respectively, where $State_{max}$>$State_{min}$. It is possible to express the number of states of each cat, such as X_i, for tracking based on the merit of a cat in the form of Equation (11) [32]:

$$\begin{aligned} &\mathrm{State(X_i)} \\ &= (\frac{\mathrm{(f(X_i) - f(w))}}{\mathrm{f(b) - f(w)}})^{\mathrm{p}} (\mathrm{State(b} \\ &- \mathrm{State(w)) + State(w)} \end{aligned} \quad 11$$

$State(X_i)$ is the number of states that a cat can search based on, and increasing the merit of each cat increases this parameter, and on the other hand, p is the power parameter, which is a number that can be He chose it between -2 and +2. Decreasing the search radius makes the search in the first iterations more exploratory and in the last iterations more exploitative. By determining the number of states to be searched by each cat, different states can be distributed around the current solution and transferred to one of the desired positions with the probability used in Equation (12) [32]:

$$X_i^{new} = X_i \pm rand.R(t) \quad 12$$

In this equation, R(1) equals the search radius in the first iteration, and five is considered. MaxIt and t are the maximum iterations and the current iteration number of the improved cat algorithm, respectively. Cats pay special attention to the position of the prey or the optimal solution and the center of gravity of the cat's gathering. In the proposed algorithm, the space between the current solution is searched from the average, as well as the space between the current solution and the optimal solution according to Equation (13) [32]:

$$\begin{aligned} X_i^{new} = X_i + r_1.(X_i - X_M) \\ + r_2.(X_i \\ - X^*) \end{aligned} \quad 13$$

X^* is the position of the most optimal solution or bait position. X_M is the average of the population of solutions. r_1 and r_2 are two random numbers between zero and one. Equation (14) is used to calculate the average solutions[32]:

$$X_M = \frac{1}{N} \sum_{i=1}^{N} X_i \quad 14$$

A better approach is to calculate the weight coefficient and importance of each solution in equation (14), and for this purpose, Equation (15) is used[32]:

$$X_M = \frac{\sum_{i=1}^{N} w_i . X_i}{N} = \frac{w_1 . X_1 + w_2 . X_2 + \cdots + w_N . X_N}{N} \quad 15$$

w_i is the fitness weight of a solution or cat, which is calculated according to Equation (16) [32]:

$$w_i = \frac{\left(f(X_i) - f(w)\right)}{f(b) - f(w)} \quad 16$$

If Equation (16) is inserted in Equation (15), then Equation (17) is obtained[32]:

$$X_M = \frac{1}{N} \sum_{i=1}^{N} \frac{\left(f(X_i) - f(w)\right)}{f(b) - f(w)} . X_i \quad 17$$

In the CHO algorithm, the learning of cats is considered. A cat like X_i randomly considers a cat like X_j, and if the merit of X_j is more than X_i, then X_i can move in the direction of the cat X_j, and this modeling is shown in Equation (18) [32]:

$$X_i^{new} = X_i + \sin\left(\frac{\pi}{2} \times \frac{t}{MaxIt}\right) \times r \times \left(X_j - X_i\right) \quad 18$$

In this equation, $\sin(\frac{\pi}{2} \times \frac{t}{MaxIt})$ is a convergence coefficient that increases the effect of the factor $X_j - X_i$ over time and the iteration of the proposed algorithm. This coefficient makes the X_i cat move towards the X_j cat faster in the final iteration and increases the speed of changing the nature of the search from global to local.

Like the cat optimization algorithm, the attack phase is run by the velocity vector in the proposed algorithm. Still, in the proposed algorithm, the velocity vector with two components is used in line with the most optimal member and the most optimal position that a cat has obtained. Equation (19) is used to define the velocity vector. In calculating the velocity vector, the most optimal position of the cats and the most optimal solution obtained by a cat so far are used to determine the direction of the velocity. c_1 and c_2 are the learning coefficients in the PSO algorithm and r_1 and $r_2$are two random numbers between zero and one[32]:

$$V_i^{new} = \omega . V_i + c_1 . r_1 . (X^* - X_i) + c_2 . r_2 . (X^b - X_i) \quad 19$$

The inertia coefficient for velocity vectors in repetition t with ω(t) and, according to Equation (20), regularly decreases according to repetition[32]:

$$\omega(t) = \omega(1) . exp\left(\frac{1-t}{MaxIt - t}\right) \quad 20$$

The initial value of the inertia coefficient is represented by ω(1), and MaxIt represents the maximum iteration of the feature selection algorithm. Feature vectors have zero and one element, and their values must remain binary even with updates. When the feature vectors are updated, their values decimalize, and conversion functions are used to make them binary again. Transformation functions such as S and V have a range of [0,1]. With transformation functions, the feature vectors normalize between zero and one. If a feature vector component has a value less than 0.5 after being affected by transformation functions such as S and V, it becomes equal to one. Otherwise, if the normalized value of a feature vector component is greater than or equal to 0.5, then the feature vector component is set equal to one. In the proposed method, the feature vectors code is first used as a member of the CHO algorithm, and an initial population of feature vectors is randomized. The feature vectors are updated with the equations of the CHO algorithm as follows(Figure 3):

- In each iteration, the worst and the best solutions are selected, and based on these two solutions, the weight of the importance of each solution or feature vector is determined.
- The number of solutions and tracking mode for each feature vector is determined based on its importance weight.
- Calculate the probability of moving to states.
- Update feature vectors by moving into the tracking phase.
- Update feature vectors by searching between mean and optimal space.
- Update feature vectors based on the velocity vector.
- Binaryizing the feature vectors and repeating the steps of the proposed algorithm to extract the optimal feature vector.

3.5. *Classification of projects with neural network VGG16*

CNN neural network has several architectures; one of the successful architectures is the VGG16 architecture. VGG16 neural network is used in most cases for image processing, and its input is images. VGG16 network is used to classify images. In the research [56], he used special coding to classify non-image samples. This research converts numerical data and features into color images, and a neural network is used for classification. The proposed method selects M samples of software defects from the dataset and considers k-selected features. In this case, the image is M*M. k columns of the desired image are the selected features of the dataset, and the rest of the columns, whose number is M-k numbers, have a value of zero. For the input to be color images, the first M sample of the software defect is taken from the R color channel, the following M sample from the G color channel, and the other M sample from the B color channel. There are two classes of color images. Their first class is software projects with defects, and the second class is successful projects. The values of each matrix are normalized between 0 and 255 so that each matrix has typical light intensity values. These images are used to train the VGG16 neural network. The input of neural network VGG16 is an example of software projects in two classes: defect and success. The output of the VGG16 neural network is two classes of success or defect of the software project.

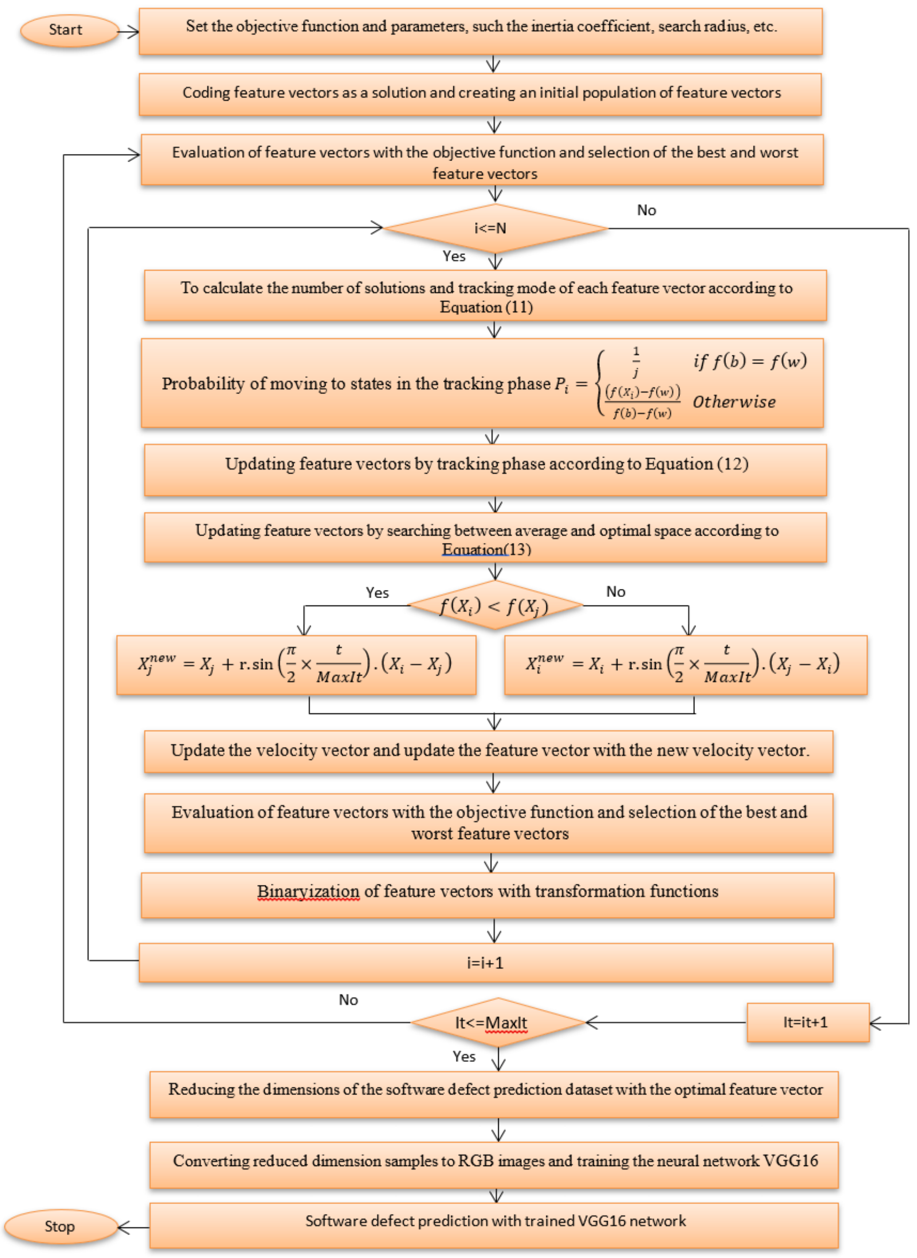

Start
Set the objective function and parameters, such the inertia coefficient, search radius, etc.
Coding feature vectors as a solution and creating an initial population of feature vectors
Evaluation of feature vectors with the objective function and selection of the best and worst feature vectors
i<=N
No
Yes
To calculate the number of solutions and tracking mode of each feature vector according to Equation (11)
Probability of moving to states in the tracking phase $P_i = \begin{cases} \frac{1}{j} & if\ f(b) = f(w) \\ \frac{(f(X_i) - f(w))}{f(b) - f(w)} & Otherwise \end{cases}$
Updating feature vectors by tracking phase according to Equation (12)
Updating feature vectors by searching between average and optimal space according to Equation(13)
Yes
$f(X_i) < f(X_j)$
No
$X_j^{new} = X_j + r.\sin\left(\frac{\pi}{2} \times \frac{t}{MaxIt}\right).(X_i - X_j)$
$X_i^{new} = X_i + r.\sin\left(\frac{\pi}{2} \times \frac{t}{MaxIt}\right).(X_j - X_i)$
Update the velocity vector and update the feature vector with the new velocity vector.
Evaluation of feature vectors with the objective function and selection of the best and worst feature vectors
Binaryization of feature vectors with transformation functions
i=i+1
No
It<=MaxIt
It=it+1
Yes
Reducing the dimensions of the software defect prediction dataset with the optimal feature vector
Converting reduced dimension samples to RGB images and training the neural network VGG16
Stop
Software defect prediction with trained VGG16 network

Figure 3. Flowchart for the proposed method

4. Experimentation

In this section, the proposed method for software defect prediction is implemented and compared with similar methods. MATLAB 2021 software is used to implement the proposed method in the feature selection phase, and Python is used for deep learning. TensorFlow and Keras libraries are used. The number of feature vectors is equal to 10, and the number of iterations of the meta-heuristic algorithm is equal to 50. The number of tests is set to 30, and the average of the trials is calculated. The normalization limit is between 0 and 1. The proposed method is compared with meta-heuristic algorithms JSO[57], AVOA[58], COA[59], WOA[60] and HHO[61]. In the JSO algorithm, the beta coefficient is equal to 3, and the omega coefficient is equal to 0.1. The coefficient C in the WOA algorithm is a random number between 0 and 2, the value of b is equal to 1, and the value of l is equal to 0.1. The value of J in the HHO algorithm is equal to 2, and E in the HHO algorithm is equal to 2. COA and AVOA parameters are initialized according to sources [58, 59].

4.1. Dataset

In this manuscript, several datasets use for evaluation. One of the datasets used to evaluate the selected algorithms is the PROMISE dataset [62]. The PROMISE dataset is one of the most widely used repositories for predicting software defects. Table (2) shows the selected data set with the number of samples and the distribution of defect classes. In Table (2), the data set from the PROMISE repository is used by NASA, which includes data sets KC1, PC5, MC1, JM1, PC1, MW1, PC2, KC3, PC4, CM1, and MC2[63].

Table 2. NASA datasets for evaluating the proposed method

Number of Features	Number of Instances	Datasets	S. No.
39	1988	MC1	1
40	125	MC2	2
38	253	MW1	3
38	705	PC1	4
37	745	PC2	5
38	1077	PC3	6
38	1287	PC4	7
39	1711	PC5	8
38	327	CM1	9
22	1183	KC1	10
40	194	KC3	11
38	327	CM1	12

In addition to the PROMISE data set, the data set, including experimental reconstruction events of four open-source software systems (JUnit et al. and ANTLR4), is used to evaluate the proposed method. The data set is available in the PROMISE repository. Table (3) [64] shows the studied features of the data set.

Table 3. List of datasets of experimental reconstruction events in four source software systems

Percentage (%)	No. of Refactoring	Instances	No. of Attributes	Dataset
5.2	23	436	134	Antlr4
1.3	9	657	134	Junit
0.9	4	439	134	MapDB
0.99	3	301	134	McMMO

4.2. Evaluation metrics

The evaluation indices according to Equations (21), (22), (23), (24) for evaluating the proposed method are used:

$$Accuracy = ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (21)$$

$$AUC = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) \quad \textbf{22}$$

$$Sensitivity = \frac{TP}{TP+FN} \quad \textbf{23}$$

$$Precision = \frac{TP}{TP+FP} \quad \textbf{24}$$

TP, TN, FP, and FN parameters are defined as follows to calculate precision, sensitivity, and precision:

- TP: A software project has a defect, and the proposed method has classified it in the defect class.
- FP: A software project has been successful, and the proposed method has classified it in the defect class.
- TN: a software project has been met with success, and the proposed method has classified it in the successful class.
- FN: A software project has a defect, and the proposed method has classified it in the successful class.

4.3. Evaluation results

In this section, the proposed method or CHO is evaluated and analyzed. In the first part of the tests, three versions of the proposed method, CHO1, CHO2, and CHO2, are developed and implemented. In the CHO1 version, only feature selection is made with the CHO algorithm. In addition to feature selection with the CHO algorithm, GAN balancing is used in the CHO2 version. In addition to feature selection with the CHO algorithm, GAN and SMOTE balancing are used in the CHO3 version. Tables (4), (5), and (6), show the accuracy(acc), sensitivity(recall), and precision of the GCV or proposed method is compared in 3 versions of the proposed method. The conducted tests show that if methods of balancing the data set increase minority samples, then the proposed method's accuracy, sensitivity, and precision will increase. The GAN+SMOTE balancing method is more effective than the GAN balancing method and increases the accuracy of the proposed method in software failure prediction. If the proposed method uses the feature selection method without balancing the dataset, it has accuracy, sensitivity, and precision of 90.98%, 89.69%, and 89.02%, respectively. If the proposed method uses the GAN balancing method, it has accuracy, sensitivity, and precision of 94.14%, 92.68%, and 92.19%, respectively. The proposed method, combining the GAN and SMOTE methods, has more accuracy, precision, and sensitivity in predicting software failure. In the optimal state, the proposed method's accuracy, sensitivity, and precision for predicting software failure are 96.69%, 96.32%, and 96.13%, respectively.

Table 4. Evaluation of the proposed method in accuracy index in the NASA dataset

Dataset	CHO1	CHO2	CHO3
PC1	95.28	96.54	98.64
PC2	97.98	98.59	99.68
PC3	89.86	94.23	95.66
PC4	94.67	96.92	98.74
PC5	78.63	84.69	92.67
JM1	83.99	92.64	95.39
KC1	82.64	86.91	93.27
KC3	92.36	94.51	96.73
CM1	91.68	93.39	95.57
MC1	97.09	98.64	99.24
MC2	95.71	97.52	98.37
MW1	91.88	95.12	96.35

Table 5. Evaluation of the proposed method in the sensitivity index in the NASA dataset

Dataset	CHO1	CHO2	CHO3
PC1	94.46	95.51	98.56
PC2	97.62	98.48	98.86
PC3	88.82	93.37	94.41
PC4	93.06	94.26	98.25
PC5	77.61	82.29	92.57
JM1	82.29	92.39	95.42
KC1	81.06	84.19	93.21
KC3	90.55	92.68	96.46
CM1	90.23	91.06	94.92
MC1	96.61	98.59	99.12
MC2	94.09	95.21	98.17
MW1	89.88	94.22	95.97

Table 6. Evaluation of the proposed method in the precision index in the NASA dataset

Dataset	CHO1	CHO2	CHO3
PC1	94.55	95.43	98.41
PC2	97.51	98.38	98.74
PC3	87.62	92.86	94.33
PC4	92.86	94.11	98.16
PC5	76.49	81.79	92.34
JM1	81.08	90.27	95.32
KC1	80.28	82.92	92.83
KC3	90.16	92.18	96.29
CM1	88.24	90.44	94.61
MC1	95.59	98.04	98.83
MC2	93.89	95.39	98.11

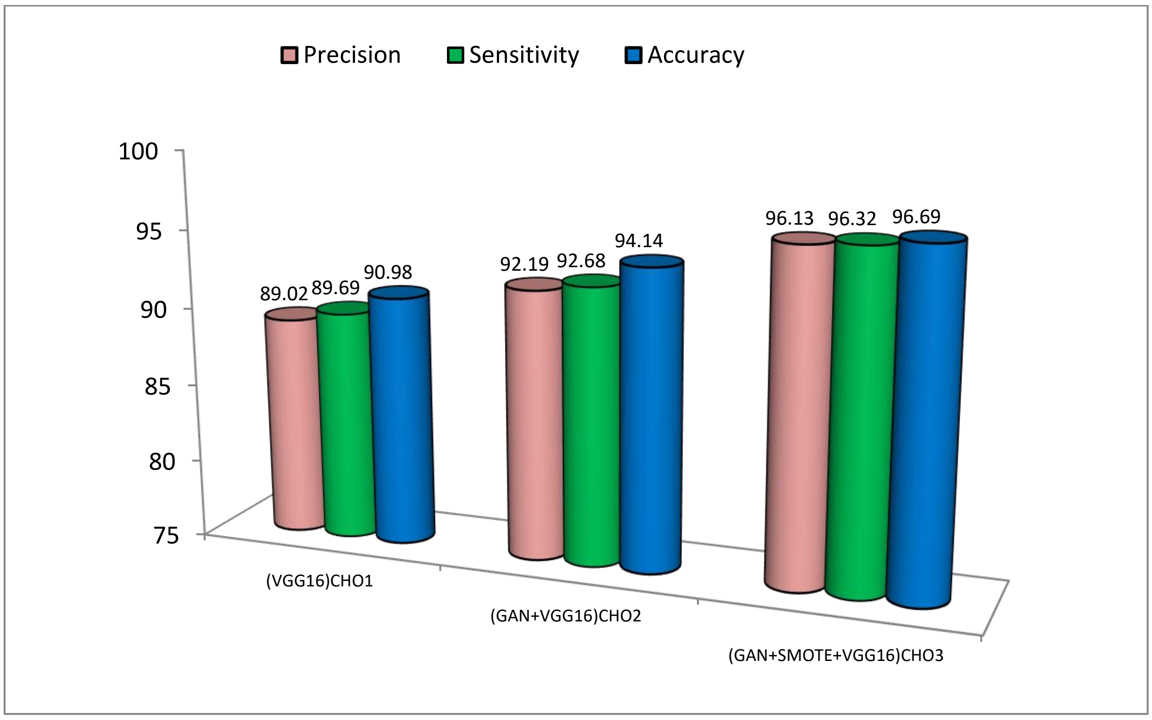


Figure 4. Evaluation of the proposed method in three modes (A)

The proposed method in the feature selection phase is based on the swarm intelligence algorithm. For the detailed analysis of the proposed method in Figure 5, the proposed method in the NASA dataset is compared with JSO, AVOA, COA, WOA, and HHO methods in 12 datasets.

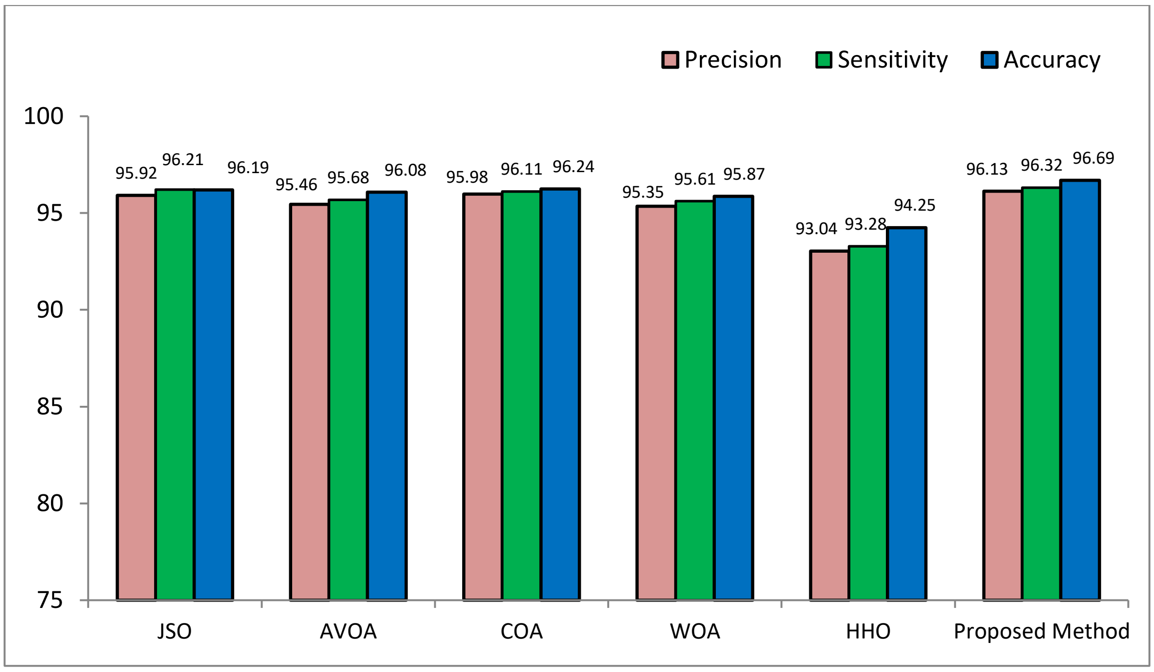


Figure 5. Evaluation of the proposed method in three modes (B)

The experiment and comparisons show that the accuracy of software defect prediction in JSO, AVOA, COA, WOA, and HHO methods is 96.19%, 96.08%, 96.24%, 95.87%, and 94.25%, respectively. The accuracy of the proposed method is 96.69%.

The proposed method (GCV) has the highest accuracy in software defect prediction among the compared methods.

The worst algorithm for predicting software failure in terms of accuracy index is the HHO algorithm. The sensitivity index in the proposed method is equal to 96.32%, and the highest sensitivity index in predicting software failure belongs to the proposed method. In the precision index, the proposed method is more successful in predicting software failure than the JSO, AVOA, COA, WOA, and HHO methods. The proposed method (GCV) performs better than other feature selection algorithms in three indicators: accuracy, sensitivity, and precision. Among the compared algorithms, COA and JSO algorithms perform better than AVOA, WOA, and HHO algorithms. The results of experiments in the CM1, JM1, and KC1 datasets are compared with the results of [62] to evaluate the proposed method accurately. In Figure 6 and Figure 7, respectively, the accuracy index and standard deviation of the tests of the proposed method are compared with K2, Hill Climbing, TAN, Decision Tree, and Random Forest methods in [62].

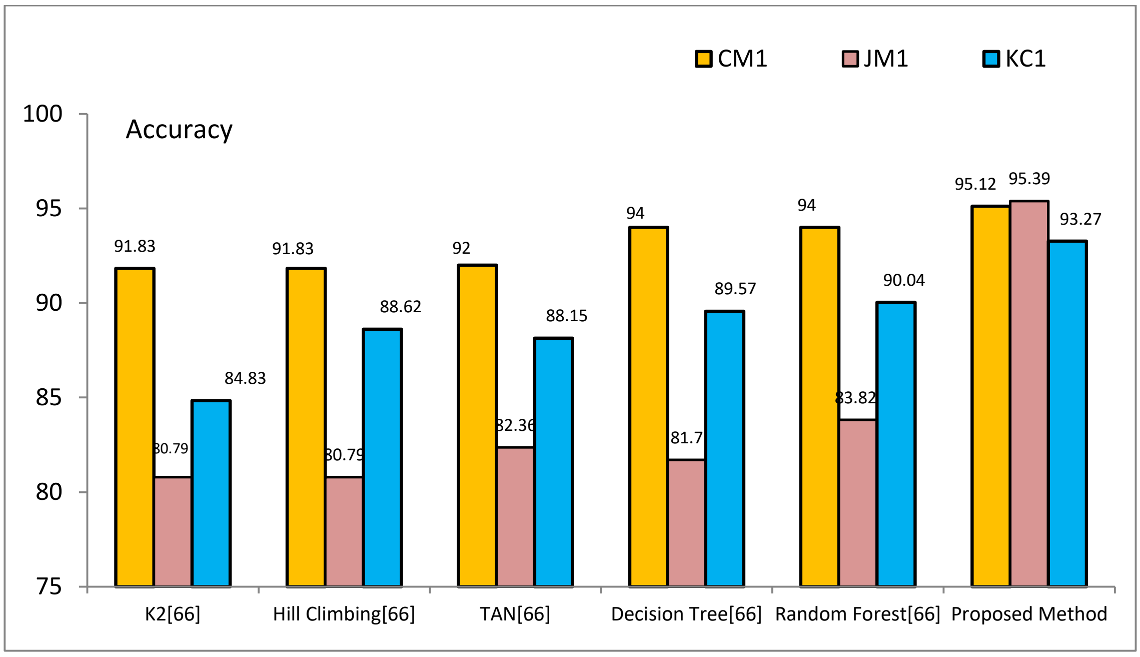


Figure 6. Comparison of the accuracy of the proposed method in software defect prediction in three datasets

In KC1, JM1, and CM1 data sets, the accuracy of the proposed method in predicting software failure is 93.27%, 95.39%, and 95.12%, respectively. Among the compared data sets, the proposed method in the JM1 data set has the highest accuracy in software failure prediction. In the KC1 data set, the random forest method has the highest prediction accuracy after the proposed method. The K2 method has the lowest accuracy in the KC1 data set. The K2 method must perform better in the software defect prediction of the three datasets. A primary index for measuring the stability of algorithms in software defect prediction is the standard deviation of tests. Figure 7 compares the proposed method with K2, Hill Climbing, TAN, Decision Tree, and Random Forest methods in the standard deviation index.

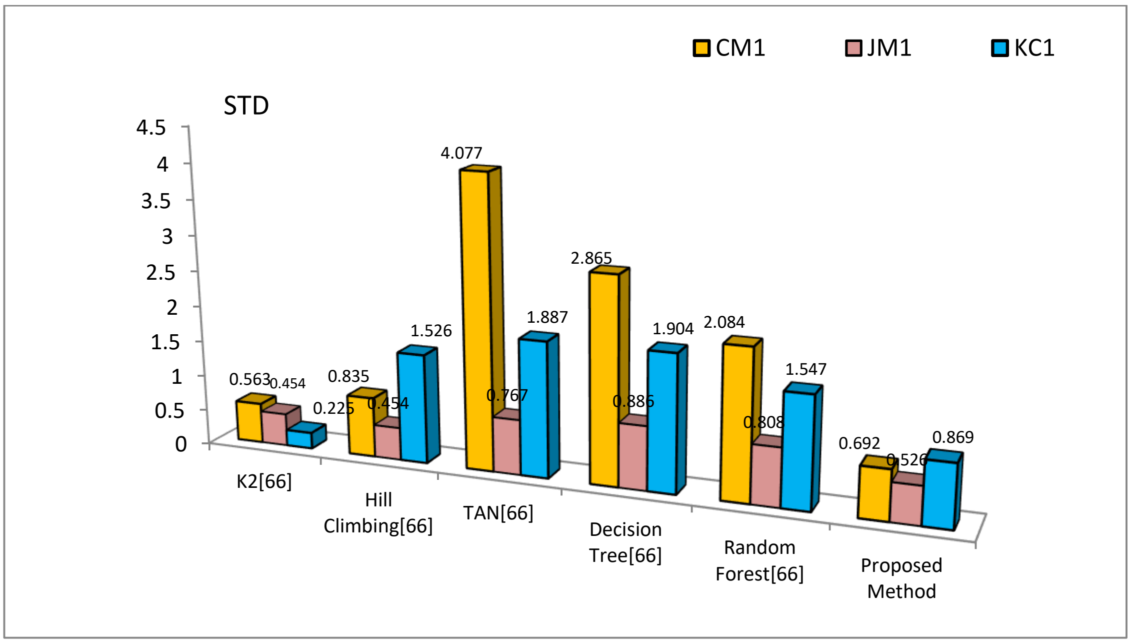


Figure 7. Comparison of the standard deviation of the proposed method in software defect prediction

Experiments and comparisons showed that the K2 method has the lowest standard deviation(STD) in the three datasets, and the proposed method ranks second in the standard deviation index. In other words, the proposed method is more accurate for software defect prediction than Hill Climbing, TAN, Decision Tree, and Random Forest methods. In Figure 8, the proposed method is compared with the results of [20] in predicting software failure in the index.

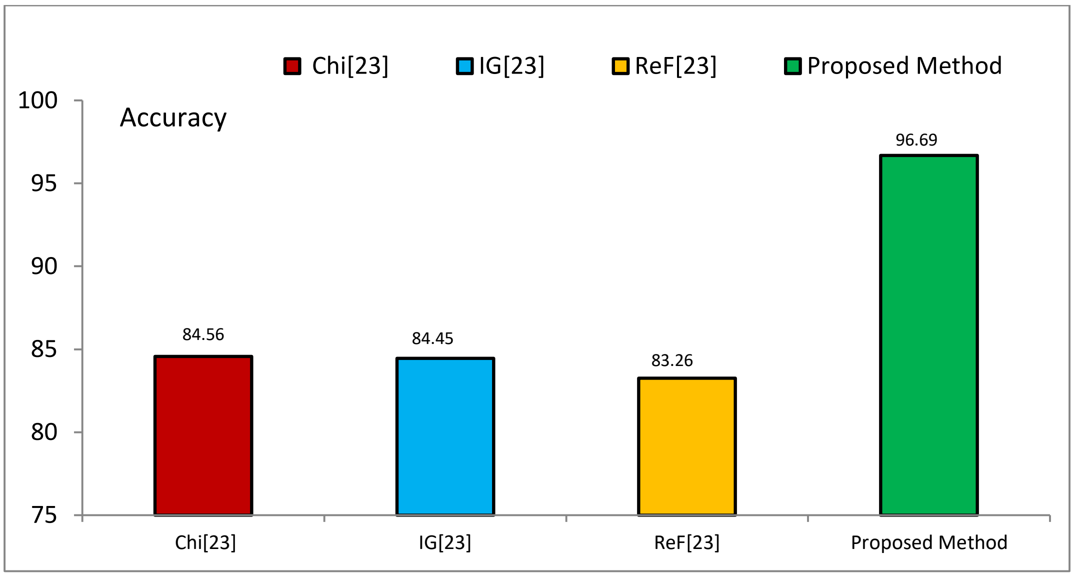


Figure 8. Comparison of the accuracy of the proposed method in software defect prediction

The proposed method in the NASA dataset has an accuracy of 96.69%, and the Chi, IG, and ReF feature selection methods have the accuracy of software defect prediction. The reason for the higher accuracy of the proposed method compared to Chi, IG, and ReF feature selection methods is the balancing of the dataset by the GAN+SMOTE method and the increase of minority samples.

Another reason is that the VGG19 classifier is more accurate than the decision tree classifier of Chi, IG, and ReF methods. The third reason is that the feature selection algorithm in the proposed method (GCV) is based on swarm intelligence and is a more optimal version of the cat optimization algorithm. However, Chi, IG, and ReF feature selection methods lack swarm intelligence mechanisms and select features based on statistical rules. The proposed method is compared with several deep learning methods for further evaluation with three software defect prediction datasets in [65]. The proposed method for comparison is implemented on three data sets, SFP XP-TDD, Eclipse, and Active MQ, and compared with CNN, LSTM, and BiLSTM methods in the accuracy index (Figure 9).

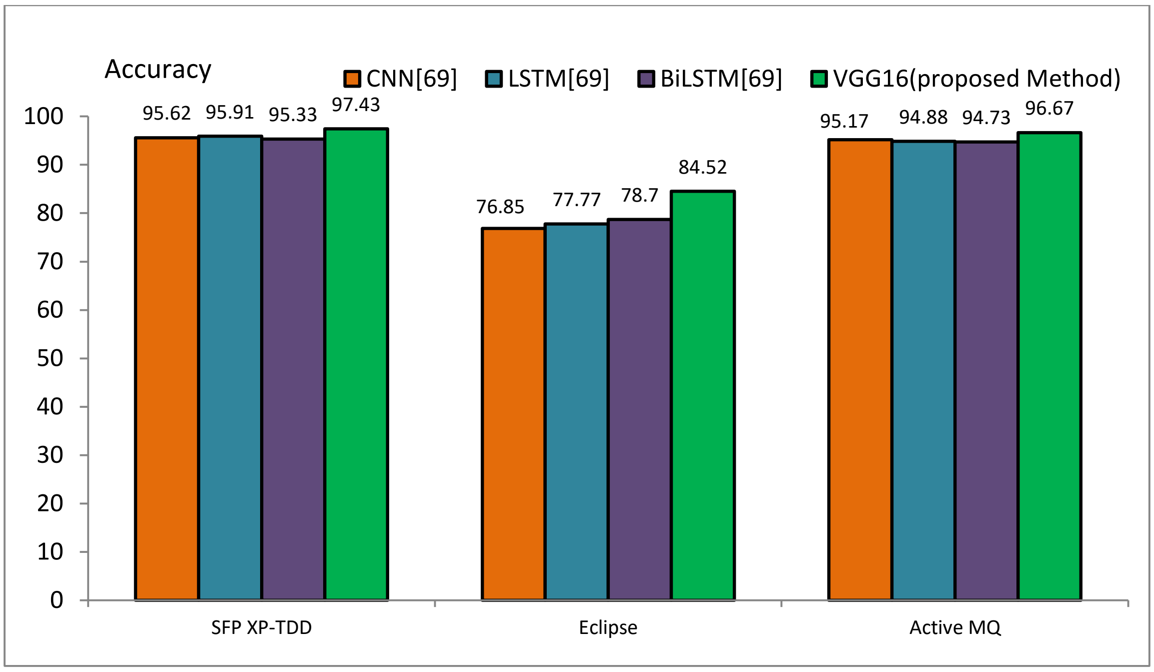


Figure 9. Comparing the accuracy of the proposed method in software defect prediction with deep learning methods

In the SFP XP-TDD data set, the accuracy of CNN, LSTM, and BiLSTM methods is 95.62%, 95.91%, and 95.33%, respectively, and the accuracy of the proposed method is 97.43%. The proposed method with VGG16 architecture is more accurate than CNN, LSTM, and BiLSTM in predicting software failure. The proposed method in the Eclipse dataset has an accuracy of about 84.52%. The accuracy of CNN, LSTM, and BiLSTM in this dataset is 76.85%, 77.77%, and 78.7%, respectively, and the proposed method is more accurate than these three deep learning methods. In the Active MQ dataset, the proposed method in software defect prediction is 96.67%, and the accuracy of CNN, LSTM, and BiLSTM methods is 95.17%, 94.88%, and 94.73%, respectively. The proposed method is more accurate than CNN, LSTM, and BiLSTM deep learning methods in the SFP XP-TDD, Eclipse, and Active MQ datasets in predicting software failure. Figure 10 compares the proposed method based on the time index with JSO, AVOA, COA, WOA, and HHO feature selection methods. Experiments show that due to the robust modeling of the proposed algorithm (GCV) in the feature selection phase, its execution time is slightly longer than the JSO and WOA methods. Experiments show that the software failure prediction time by the proposed method is less than the AVOA, COA, and HHO methods.

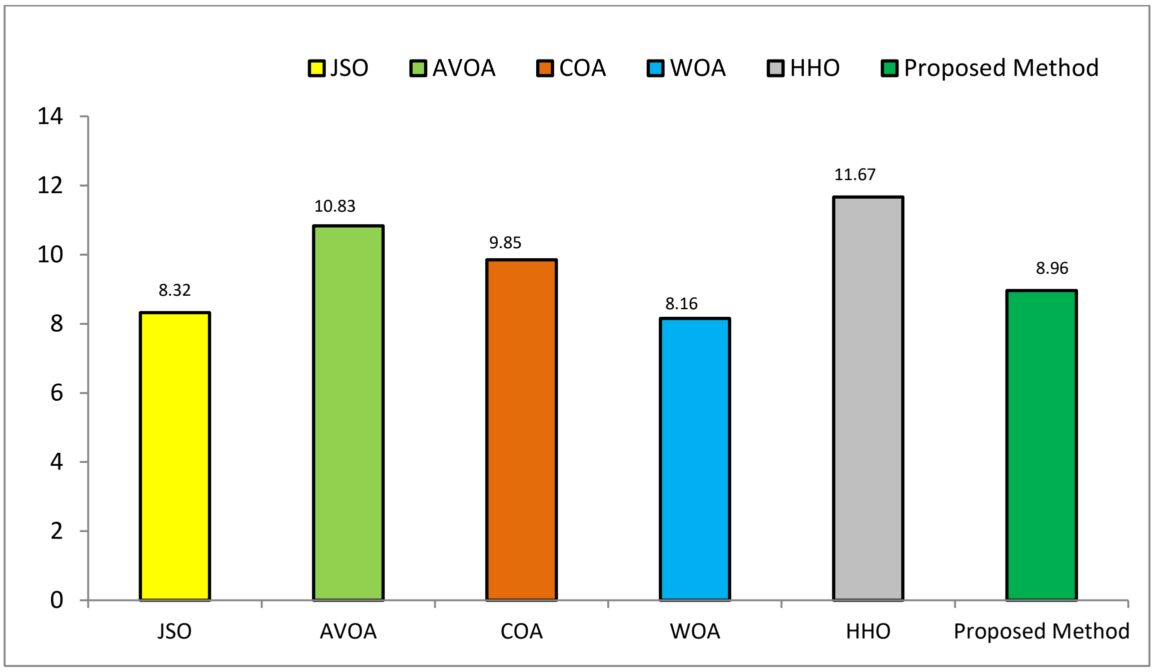


Figure 10. Comparison of execution time in software failure prediction

According to the experiments, the proposed method of predicting software failure has the following advantages:

- The proposed method (GCV) is more accurate than deep learning methods such as CNN, LSTM, and BiLSTM in predicting software failure.
- The proposed method (GCV) is more accurate due to the balancing of the data set than the forecasting methods that do not have the balancing of the data set.
- Due to intelligent feature selection, the proposed method is more accurate than other feature selection methods such as JSO, AVOA, COA, WOA, and HHO.
- The proposed method can predict software failure due to the creation of artificial data on small data sets.
- In addition to the NASA dataset, the proposed method is more accurate than deep learning methods in the SFP XP-TDD dataset, Eclipse Active MQ.

The proposed method has the following disadvantages:

- Prediction accuracy depends on the quality of training data.
- The process of teaching deep learning methods is time-consuming and requires appropriate hardware.
- The proposed method suffers from uncertainty at this stage due to using meta-heuristic algorithms in feature selection.
- Determining the parameters of the algorithms used in the proposed method requires optimization.

The problem of the proposed method and other software failure prediction methods that reduce the quality of prediction are summarized below:

- The proposed method and many software failure prediction methods cannot predict practical software failure.
- Better quality modeling is provided if software failure prediction uses human error factors.
- Predictive methods should be able to read program codes, and feature extraction is a crucial step in understanding code and finding violations.
- Using Natural Language Processing (NLP) increases the ability to predict software failure, detect faulty codes, and improve the quality of the prediction model.

5. Conclusion

The manuscript uses the Cat Hunting Optimization (CHO) algorithm and deep learning to predict a software project's failure. The advantage of the improved version of the cat algorithm is the ability to search for exploration and exploratory searches. The number of data set samples and minority classes increases to reduce the error of the software failure prediction model with the game theory based on the GAN neural network improved with SMOTE. The binary version of the CHO algorithm selects the essential features of software projects that play an important role in predicting software failure. The selected optimal features of the data set convert into RGB color images, and the images are set as the input of the VGG 16 deep learning network. The role of the convolutional neural network is to classify the examples of software projects into two classes: failure and success. The experiments performed on the NASA software project dataset show that the proposed method in predicting the failure of software projects has an accuracy, sensitivity, and precision of 96.69%, 96.32%, and 96.13%, respectively. Without balancing the data set, the proposed method has an accuracy, sensitivity, and precision of 90.98%, 89.69%, and 89.02%, respectively. Balancing the data set with the GAN and SMOTE neural network increases the accuracy of the proposed failure prediction model. The proposed method accurately predicts software failure from neural networks such as MLP, RNN, LSTM, and Bi-LSTM. The proposed is more accurate in predicting software failure than WOA, HHO, AVOA, JSO, and COA algorithms.

The main advantage of the proposed method (GCV) is balancing the data set and increasing the examples of software projects with precise methods based on game theory. Another advantage of the proposed method is the intelligent selection of features of software projects in predicting software failure. Another advantage of the proposed method (GCV) is the ability to generate artificial samples by combining GAN and SMOTE. The proposed method is more accurate than some deep learning and machine learning methods for predicting the failure of software projects. In addition to the mentioned advantages, the proposed method has several challenges. The disadvantages of the proposed method are the prediction model's complexity and the uncertainty of meta-heuristic algorithms in feature selection. In future work, a hybrid neural network based on CNN-LSTM architecture will be used to classify software projects into failure and success. Another future work is extracting features from the programming code of software projects with natural processing language (NLP) and pre-trained BERT networks.

Authors' Contributions

Authors equally contributed to this article.

Data Availability Statement

The datasets generated during and/or analysed during the current study are available in http://promise.site.uottawa.ca/SERepository/datasets-page.html

Acknowledgments

Authors thank all participants who participate in this study.

Declaration of Interest

The authors report no conflict of interest.

Funding

According to the authors, this article has no financial support.

Ethical Considerations

All procedures performed in this study were under the ethical standards.

References

[1] R. Malhotra, S. Chawla, and A. Sharma, "Software defect prediction using hybrid techniques: a systematic literature review," *Soft Computing*, pp. 1-34, 2023, doi: 10.1007/s00500-022-07738-w.

[2] D. A. Rebro, B. Rossi, and S. Chren, "Source Code Metrics for Software Defects Prediction," *arXiv preprint arXiv:2301.08022*, 2023, doi: 10.1145/3555776.3577809.

[3] R. Vashisht and S. A. M. Rizvi, "Addressing Noise and Class Imbalance Problems in Heterogeneous Cross-Project Defect Prediction: An Empirical Study," *International Journal of e-Collaboration (IJeC)*, vol. 19, no. 1, pp. 1-27, 2023, doi: 10.4018/IJeC.315777.

[4] F. Huang and L. Strigini, "HEDF: A Method for Early Forecasting Software Defects based on Human Error Mechanisms," *IEEE Access*, 2023, doi: 10.1109/ACCESS.2023.3234490.

[5] N. Yadav and V. Yadav, "Software reliability prediction and optimization using machine learning algorithms: A review," *Journal of Integrated Science and Technology*, vol. 11, no. 1, pp. 457-457, 2023.

[6] R. Moussa and F. Sarro, "On the use of evaluation measures for defect prediction studies," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, July 2022, pp. 101-113, doi: 10.1145/3533767.3534405.

[7] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools," *Engineering Applications of Artificial Intelligence*, vol. 111, p. 104773, 2022, doi: 10.1016/j.engappai.2022.104773.

[8] G. G. Cabral and L. L. Minku, "Towards reliable online just-in-time software defect prediction," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 1342-1358, 2022, doi: 10.1109/TSE.2022.3175789.

[9] B. U. Sharma and R. Sadam, "Do the Defect Prediction Models Really Work?," *arXiv e-prints*, 2023.

[10] M. Gupta, K. Rajnish, and V. Bhattacharya, "Effectiveness of Ensemble Classifier Over State-Of-Art Machine Learning Classifiers for Predicting Software Faults in Software Modules," in *Machine Learning, Image Processing, Network Security and Data Sciences: Select Proceedings of 3rd International Conference on MIND 2021*, January 2023, pp. 77-88, doi: 10.1007/978-981-19-5868-7_7.

[11] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Machine learning-based software defect prediction for mobile applications: A systematic literature review," *Sensors*, vol. 22, no. 7, p. 2551, 2022, doi: 10.3390/s22072551.

[12] S. Gurung, "Performing Software Defect Prediction Using Deep Learning," in *Cognition and Recognition: 8th International Conference, ICCR 2021*, January 2023, pp. 319-331, doi: 10.1007/978-3-031-22405-8_25.

[13] N. C. Shrikanth, S. Majumder, and T. Menzies, "Early life cycle software defect prediction. why? how?," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021, pp. 448-459, doi: 10.1109/ICSE43902.2021.00050.

[14] M. Shafiq, F. H. Alghamedy, N. Jamal, T. Kamal, Y. I. Daradkeh, and M. Shabaz, "Scientific programming using optimized machine learning techniques for software fault prediction to improve software quality," *IET Software*, 2023, doi: 10.1049/sfw2.12091.

[15] S. Hameed, Y. Elsheikh, and M. Azzeh, "An optimized case-based software project effort estimation using genetic algorithm," *Information and Software Technology*, vol. 153, p. 107088, 2023, doi: 10.1016/j.infsof.2022.107088.

[16] F. Meng, W. Cheng, and J. Wang, "Semi-supervised software defect prediction model based on tri-training," *KSII Transactions on Internet & Information Systems*, vol. 15, no. 11, 2021, doi: 10.3837/tiis.2021.11.009.

[17] H. Krasner, "The cost of poor software quality in the US: A 2020 report," *Proc. Consortium Inf. Softw. QualityTM (CISQTM)*, 2021.

[18] M. Jagtap, P. Katragadda, and P. Satelkar, "Software Reliability: Development of Software Defect Prediction Models Using Advanced Techniques," in *2022 Annual Reliability and Maintainability Symposium (RAMS)*, January 2022, pp. 1-7, doi: 10.1109/RAMS51457.2022.9893986.

[19] M. Azzeh, Y. Elsheikh, A. B. Nassif, and L. Angelis, "Examining the performance of kernel methods for software defect prediction based on support vector machine," *Science of Computer Programming*, vol. 226, p. 102916, 2023, doi: 10.1016/j.scico.2022.102916.

[20] A. Iqbal and S. Aftab, "A Classification Framework for Software Defect Prediction Using Multi-filter Feature Selection Technique and MLP," *International Journal of Modern Education & Computer Science*, vol. 12, no. 1, 2020, doi: 10.5815/ijmecs.2020.01.03.

[21] Z. Marian, I. G. Mircea, I. G. Czibula, and G. Czibula, "A novel approach for software defect prediction using fuzzy decision trees," in *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, September 2016, pp. 240-247, doi: 10.1109/SYNASC.2016.046.

[22] A. Balaram and S. Vasundra, "Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm," *Automated Software Engineering*, vol. 29, no. 1, p. 6, 2022, doi: 10.1007/s10515-021-00311-z.

[23] J. Goyal and R. Ranjan Sinha, "Software defect-based prediction using logistic regression: Review and challenges," in *Second International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2021*, 2022, pp. 233-248, doi: 10.1007/978-981-16-4641-6_20.

[24] C. Pornprasit and C. K. Tantithamthavorn, "Deeplinedp: Towards a deep learning approach for line-level defect prediction," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 84-98, 2022, doi: 10.1109/TSE.2022.3144348.

[25] Z. M. Zain, S. Sakri, N. H. A. Ismail, and R. M. Parizi, "Software defect prediction harnessing on multi 1-dimensional convolutional neural network structure," *Computers, Materials and Continua*, vol. 71, no. 1, p. 1521, 2022, doi: 10.32604/cmc.2022.022085.

[26] M. N. Uddin, B. Li, Z. Ali, P. Kefalas, I. Khan, and I. Zada, "Software defect prediction employing BiLSTM and BERT-based semantic feature," *Soft Computing*, vol. 26, no. 16, pp. 7877-7891, 2022, doi: 10.1007/s00500-022-06830-5.

[27] W. Zheng, L. Tan, and C. Liu, "Software Defect Prediction Method Based on Transformer Model," in *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, June 2021, pp. 670-674, doi: 10.1109/ICAICA52286.2021.9498179.

[28] B. J. Odejide *et al.*, "An Empirical Study on Data Sampling Methods in Addressing Class Imbalance Problem in Software Defect Prediction," in *Software Engineering Perspectives in Systems: Proceedings of 11th Computer Science On-line Conference 2022*, July 2022, vol. 1, pp. 594-610, doi: 10.1007/978-3-031-09070-7_49.

[29] A. O. Balogun *et al.*, "Impact of feature selection methods on the predictive performance of software defect prediction models: an extensive empirical study," *Symmetry*, vol. 12, no. 7, p. 1147, 2020, doi: 10.3390/sym12071147.

[30] S. Zhang, S. Jiang, and Y. Yan, "A Software Defect Prediction Approach Based on BiGAN Anomaly Detection," *Scientific Programming*, 2022, doi: 10.1155/2022/5024399.

[31] S. Feng, J. Keung, P. Zhang, Y. Xiao, and M. Zhang, "The impact of the distance metric and measure on SMOTE-based techniques in software defect prediction," *Information and Software Technology*, vol. 142, p. 106742, 2022, doi: 10.1016/j.infsof.2021.106742.

[32] A. Ghaedi, A. K. Bardsiri, and M. J. Shahbazzadeh, "Cat hunting optimization algorithm: a novel optimization algorithm," *Evolutionary Intelligence*, vol. 16, no. 2, pp. 417-438, 2023, doi: 10.1007/s12065-021-00668-w.

[33] J. Sun, L. Chen, F. Cang, H. Li, and F. Pi, "Civil Aircraft Airborne Software Safety and Reliability Study Based on RTCA/DO-178C," in *Proceedings of the 5th International Conference on Computer Science and Software Engineering*, October 2022, pp. 27-33, doi: 10.1145/3569966.3569974.

[34] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, p. 111537, 2023, doi: 10.1016/j.jss.2022.111537.

[35] Y. Guan, J. Zhang, R. Zhang, J. Gan, and M. Liu, "Fault Handling of Digital System in Nuclear Power Plants," in *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, August 2022, pp. 166-171, doi: 10.1109/AEECA55500.2022.9919016.

[36] A. Abdu, Z. Zhai, R. Algabri, H. A. Abdo, K. Hamad, and M. A. Al-antari, "Deep Learning-Based Software Defect Prediction via Semantic Key Features of Source Code-

Systematic Survey," *Mathematics,* vol. 10, no. 17, p. 3120, 2022, doi: 10.3390/math10173120.

[37] Y. Tang, Q. Dai, M. Yang, T. Du, and L. Chen, "Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm," *International Journal of Machine Learning and Cybernetics,* pp. 1-21, 2023, doi: 10.1007/s13042-022-01740-2.

[38] S. Guo, J. Wang, Z. Xu, L. Huang, H. Li, and R. Chen, "Feature transfer learning by reinforcement learning for detecting software defect," *Software: Practice and Experience,* vol. 53, no. 2, pp. 366-389, 2023, doi: 10.1002/spe.3152.

[39] A. Alazba and H. Aljamaan, "Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles," *Applied Sciences,* vol. 12, no. 9, p. 4577, 2022, doi: 10.3390/app12094577.

[40] M. Cui, S. Long, Y. Jiang, and X. Na, "Research of Software Defect Prediction Model Based on Complex Network and Graph Neural Network," *Entropy,* vol. 24, no. 10, p. 1373, 2022, doi: 10.3390/e24101373.

[41] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Deep Learning-Based Defect Prediction for Mobile Applications," *Sensors,* vol. 22, no. 13, p. 4734, 2022, doi: 10.3390/s22134734.

[42] M. A. Khan *et al.*, "Software defect prediction using artificial neural networks: A systematic literature review," *Scientific Programming,* 2022, doi: 10.1155/2022/2117339.

[43] S. Goyal, "Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction," *Artificial Intelligence Review,* vol. 55, no. 3, pp. 2023-2064, 2022, doi: 10.1007/s10462-021-10044-w.

[44] S. Goyal, "Effective software defect prediction using support vector machines (SVMs)," *International Journal of System Assurance Engineering and Management,* vol. 13, no. 2, pp. 681-696, 2022, doi: 10.1007/s13198-021-01326-1.

[45] C. Arun and C. Lakshmi, "Genetic algorithm-based oversampling approach to prune the class imbalance issue in software defect prediction," *Soft Computing,* vol. 26, no. 23, pp. 12915-12931, 2022, doi: 10.1007/s00500-021-06112-6.

[46] N. Zhang, S. Ying, K. Zhu, and D. Zhu, "Software defect prediction based on stacked sparse denoising autoencoders and enhanced extreme learning machine," *IET Software,* vol. 16, no. 1, pp. 29-47, 2022, doi: 10.1049/sfw2.12029.

[47] R. A. Khurma, H. Alsawalqah, I. Aljarah, M. A. Elaziz, and R. Damaševičius, "An enhanced evolutionary software defect prediction method using island moth flame optimization," *Mathematics,* vol. 9, no. 15, p. 1722, 2021, doi: 10.3390/math9151722.

[48] L. Yang, Z. Li, D. Wang, H. Miao, and Z. Wang, "Software defects prediction based on hybrid particle swarm optimization and sparrow search algorithm," *IEEE Access,* vol. 9, pp. 60865-60879, 2021, doi: 10.1109/ACCESS.2021.3072993.

[49] T. Ye, W. Li, J. Zhang, and Z. Cui, "A novel multi-objective immune optimization algorithm for under sampling software defect prediction problem," *Concurrency and Computation: Practice and Experience,* vol. 35, no. 4, p. e7525, 2023, doi: 10.1002/cpe.7525.

[50] S. Kanwar *et al.*, "Efficient Random Forest Algorithm for Multi-objective Optimization Optimized ensemble machine learning model for software bugs prediction," *Innovations in Systems and Software Engineering,* vol. 19, no. 1, pp. 91-101, 2023, doi: 10.1007/s11334-022-00506-x.

[51] S. Goyal, "3PcGE: 3-parent child-based genetic evolution for software defect prediction," *Innovations in Systems and Software Engineering,* vol. 19, no. 2, pp. 197-216, 2023, doi: 10.1007/s11334-021-00427-1.

[52] S. Goyal, "Genetic evolution-based feature selection for software defect prediction using SVMs," *Journal of Circuits, Systems and Computers,* vol. 31, no. 11, p. 2250161, 2022, doi: 10.1142/S0218126622501614.

[53] S. Goyal, "FOFS: firefly optimization for feature selection to predict fault-prone software modules In - Data Engineering for Smart Systems: Proceedings of SSIC 2021," Springer Singapore, 2022, pp. 479-487.

[54] K. L. Wong, K. S. Chou, R. Tse, S. K. Tang, and G. Pau, "A Novel Fusion Approach Consisting of GAN and State-of-Charge Estimator for Synthetic Battery Operation Data Generation," *Electronics,* vol. 12, no. 3, p. 657, 2023, doi: 10.3390/electronics12030657.

[55] J. H. Joloudari, A. Marefat, M. A. Nematollahi, S. S. Oyelere, and S. Hussain, "Effective Class-Imbalance Learning Based on SMOTE and Convolutional Neural Networks," *Applied Sciences,* vol. 13, no. 6, p. 4006, 2023, doi: 10.3390/app13064006.

[56] A. El-Ghamry, A. Darwish, and A. E. Hassanien, "An optimized CNN-based intrusion detection system for reducing risks in smart farming," *Internet of Things,* vol. 22, p. 100709, 2023, doi: 10.1016/j.iot.2023.100709.

[57] J. S. Chou and D. N. Truong, "A novel metaheuristic optimizer inspired by behavior of jellyfish in ocean," *Applied Mathematics and Computation,* vol. 389, p. 125535, 2021, doi: 10.1016/j.amc.2020.125535.

[58] B. Abdollahzadeh, F. S. Gharehchopogh, and S. Mirjalili, "African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems," *Computers & Industrial Engineering,* vol. 158, p. 107408, 2021, doi: 10.1016/j.cie.2021.107408.

[59] M. Dehghani, Z. Montazeri, E. Trojovská, and P. Trojovský, "Coati Optimization Algorithm: A new bio-inspired metaheuristic algorithm for solving optimization problems," *Knowledge-Based Systems,* vol. 259, p. 110011, 2023, doi: 10.1016/j.knosys.2022.110011.

[60] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software,* vol. 95, pp. 51-67, 2016, doi: 10.1016/j.advengsoft.2016.01.008.

[61] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Generation Computer Systems,* vol. 97, pp. 849-872, 2019, doi: 10.1016/j.future.2019.02.028.

[62] M. J. Hernández-Molinos, A. J. Sánchez-García, R. E. Barrientos-Martínez, J. C. Pérez-Arriaga, and J. O. Ocharán-Hernández, "Software Defect Prediction with Bayesian Approaches," *Mathematics,* vol. 11, no. 11, p. 2524, 2023, doi: 10.3390/math11112524.

[63] H. Das, S. Prajapati, M. K. Gourisaria, R. M. Pattanayak, A. Alameen, and M. Kolhar, "Feature Selection Using Golden Jackal Optimization for Software Fault Prediction," *Mathematics,* vol. 11, no. 11, p. 2438, 2023, doi: 10.3390/math11112438.

[64] M. Akour, M. Alenezi, and H. Alsghaier, "Software Refactoring Prediction Using SVM and Optimization Algorithms," *Processes,* vol. 10, no. 8, p. 1611, 2022, doi: 10.3390/pr10081611.

[65] E. Borandag, "Software Fault Prediction Using an RNN-Based Deep Learning Approach and Ensemble Machine Learning Techniques," *Applied Sciences,* vol. 13, no. 3, p. 1639, 2023, doi: 10.3390/app13031639.