Knowledge and Information Systems (2026) 68:20
https://doi.org/10.1007/s10115-025-02657-2

BRIEF REPORT

Enhanced software defect prediction using edge feature and self-attention GAN with pelican optimization

S. V. Gayetri Devi[1] · P. V. V. Satyanarayana[2] · A. Mani[3] · S. Praveena[4]

Received: 16 October 2023 / Revised: 25 November 2024 / Accepted: 30 December 2024

Abstract
Software defect prediction (SDP) is an essential task in software engineering for identifying defective modules early in the development process, thereby improving software quality and reducing maintenance costs. Existing SDP models often face a number of difficulties despite notable improvements, such as imbalanced datasets and the inability to capture the complex relationships within the code. In this manuscript, a software defect prediction using edge feature with self-attention-based cycle-consistent generative adversarial network optimized by the pelican optimization algorithm (SDP-ESA-CycleGAN-POA) is proposed to enhance the defect prediction by capturing semantic features within the software code. The proposed method utilizes abstract syntax tree (AS-Tree) tokens and dense vector transformation through word embedding to improve prediction accuracy. Then, edge feature and self-attention-based cycle-consistent generative adversarial network (ESA-CycleGAN) predicts the data as buggy and clean. Finally, the pelican optimization algorithm (POA) is employed to enhance the weight parameters of the ESA-CycleGAN, leading to improved defect prediction accuracy. This research evaluates the ESA-CycleGAN model using the PROMISE dataset. The proposed SDP-ESA-CycleGAN-POA method achieves 21.57%, 23.41%, 16.10% and 18.73% higher accuracy compared with the existing models: a new method to SDP accuracy with machine learning (SDP-RF-LR), SDP using optimum trained convolutional neural network (SDP-OT-CNN), SDP utilizing intelligent ensemble-based method (SDP-RF-SVM-ANN) and SDP through neural network and feature selections (SDP-RBFNN-FS) respectively.

Keywords Abstract syntax tree · Edge extraction module · Software defect prediction · Pelican optimization algorithm

✉ S. V. Gayetri Devi
gayetri.venkhatraman@gmail.com

[1] Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College Of Engineering, Chennai, Tamil Nadu, India

[2] Department of Electrical and Electronics Engineering, Malla Reddy Engineering College, Hyderabad, India

[3] Department of Computer Science and Engineering, S.A. Engineering College, Chennai, Tamil Nadu, India

[4] Department of Electronics and Communication Engineering, Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad-75, Telangana, India

Published online: 26 December 2025

1 Introduction

Over the past 20 years, the demand for software has increased significantly across a wide range of applications [1]. Each year, a significant number of software applications are created in response to the growing demand from contemporary technology-based businesses and business applications. However, software quality is frequently disregarded in the process of creation [2, 3]. Software applications have grown to be indispensable in recent years for daily and corporate usage [4]. A minor software flaw in a business organization could result in a loss for the sector and worse customer satisfaction [5]. Software testing-related problems become urgent concerns. Therefore, a software testing is done automatically, which can reduce implementation costs and increase performance [6, 7]. This issue can be avoided if the tester is informed about the general software development process and the origins of probable errors. This can aid in better planning and carrying out software development projects in addition to lowering the overall cost of software development [8]. The software development life cycle is regarded as the primary driver of software development in software industries [9, 10]. Recent data indicates that project managers view early defect prediction as an essential and difficult role expected increasingly [11]. The discipline of software development has evolved to include complex issue domains, complicated development processes, unpredictable software performance and rising software application requirements [12, 13]. Certain faults in software development are unavoidable and lead to software performance decrease with extensive documentation and a well-organized procedure [14]. SDP is a technique that utilizes a method to estimate the code areas that has defects [15]. The processing of SDP encompasses three steps: historical defect database collection; utilizing the historical data to regression training or categorization under deep learning or machine learning methods; and using the trained mode to estimate likelihood of software defects [16]. On the view of various database granularities, SDP can be separated as four levels: package, file, method and change [17]. On the view of diverse metrics, SDP is separated as two defect prediction phases: static and dynamic [18]. SDP uses static software measurements to forecast the quantity or distribution of defects [19]. The primary goal of dynamic defect prediction approaches is to use defect generation time to anticipate the distribution of system flaws over time [20].

Many researches have been presented for SDP to predict defective instances from historical data [21]. But the presented SDP methods struggle with class imbalance, where defect-free instances considerably exceed the defective ones. Traditional methods often emphasize basic statistical metrics or simple software measures, such as lines of code, complexity and historical defect data. However, these approaches fall short in capturing the intricate interactions among various features that contribute to software defects. Current methods often overlook temporal and contextual changes in software development, such as new features, code modifications or evolving developer practices, all of which can significantly impact defect prediction.

This study develops an ESA-CycleGAN model to explore how it can effectively improve defect prediction. In the proposed model, AS-Trees provide a hierarchical representation of the code's syntactic structure, allowing for a deeper understanding of the relationships between different codes constructs. By using edge features from the software's dependency graph, the proposed model can learn a richer set of features that capture the intricate relationships between different components of the software. The self-attention GAN facilitates the method to focus on the relevant parts of the software by assigning attention weights to different features. This mechanism addresses the issue of traditional models overlooking important features by dynamically learning. The pelican optimization algorithm is used to

fine-tune model parameters, helping to prevent overfitting and improve generalization by finding the optimal balance between model complexity and accuracy.

To assess the proposed SDP-ESA-CycleGAN-POA approach, the following two research questions were explored:

- How the proposed SDP-ESA-CycleGAN-POA method compare to the existing methods under accuracy for detecting buggy and clean software in software defect prediction?
- What are the advantages of with the pelican optimization algorithm (POA) in optimizing the weight parameter of ESA-CycleGAN?

The major concepts of this paper are abridged below:

- SDP-ESA-CycleGAN-POA method is proposed.
- Utilization of self-attention mechanisms to extract essential features from abstract syntax tree (AS-Tree) tokens while disregarding less relevant ones, leading to improved accuracy and efficiency in defect prediction.
- Introduction of the ESA-CycleGAN [22] model for defect prediction in software engineering, which utilizes AS-Tree tokens and dense vector transformation through word embedding to enhance accuracy in defect prediction.
- Integration of the POA [23] is to improve the weight parameters of ESA-CycleGAN method, thereby obtaining the optimal solution and enhancing the SDP.

The remaining paper is structured as: portion 2 presents the related works, portion 3 explains the proposed approach, portion 4 exemplifies the outcomes with discussions, portion 5 explains threats to validity and portion 6 gives the conclusion.

2 Related work

Several research works were suggested in the studies based on SDP utilizing deep learning; some of them are revised here.

Mehmood et al. [24] have suggested a novel method for enhancing machine learning's capacity to forecast software defects. The suggested paper uses feature selection in combination with machine learning approaches, like decision stump, random forest, Rule ZeroR, J48, Lazy IBK, support vector machine, logistic regression, multi-layer perceptron, Bayesian net and neural networks for attaining higher defect prediction accuracy without using feature selection. It provides higher accuracy and lower sensitivity.

Balasubramaniam and Gollagi [25] have suggested an optimally trained convolutional neural network for SDP. A multi-stage methodology for software defect prediction was presented. First, there was a pre-processing phase for the input data. Furthermore, enhanced principle component analysis was used to choose the required attributes. Then, a CNN (convolutional neural network) has been upgraded to be utilized for predicting flaws based on the selected features. The suggested hybrid seagull adopted ant lion optimization method optimizes the CNN weights to achieve accurate and precise detection. It attains greater precision and greater computation time.

Ali et al. [26] have suggested software defect prediction with an intelligent ensemble-based model. The presented method uses a two-step prediction procedure to identify faulty modules. During the initial phase, four supervised machine learning techniques were utilized: artificial neural network, support vector machine, random forest and naïve Bayes. To attain the maximum accuracy feasible, these algorithms undergo iterative parameter optimization. The voting ensemble that incorporates the predictive accuracy of each individual

classifier was used to create the final predictions in the second step. It provides high specificity and low accuracy.

Alkhasawneh [27] have presented feature selection and neural network-based software defect prediction. Here, suggested a methodology using feature selection and classifications to forecast the software failure. Radial basis function neural network (RBF) was utilized for classification, while a correlation-based technique was applied for feature selection. In addition, fourteen NASA datasets were used to test the technique was provided. The RBF was instructed and evaluated before and after evaluations. Four techniques are utilized to assess the efficacy of approaches with precision, recall, F-measure and accuracy was presented. It attained greater accuracy, but lesser F-measure.

Rajnish et al. [28] have suggested new convolutional neural network model for software defect prediction. TensorFlow and Keras were integrated into the Python programming language framework. Four NASA scheme databases (KC1, PC1, PC2 and KC3) chosen from the PROMISE repository were used in a comparative analysis using machine learning approaches like random forest, decision trees, naïve Bayes, DNN on F-measure, recall, precision, accuracy. It attains lesser error rate and greater computation time.

Parashar et al. [29] have introduced a machine learning method to predicting software flaws using multi-core parallel computing. The suggested work provides a multiple core parallel machine learning technique for SDP, enabling the component classification as either defective or non-defective. The presented models development, training and testing, the number of CPU cores employed for processing has changed. Using the suggested approach, several empirical investigations were conducted on 11 NASA/PROMISE software systems and other relevant sources. It reached higher accuracy and lower sensitivity.

Manchala and Bisi [30] have suggested a diversity-based imbalance learning strategy that uses machine learning models to predict software faults. The imbalance learning approach based on weighted average centroid (WACIL) was used in the presented study. The WACIL initially identifies occurrences were borderline, then uses a weighted average centroid idea to generate pseudo-data about those instances and filters out unwanted noise data using filtration procedure. In which, 24 PROMISE and NASA projects were used for the experiments and compared to some of the current sampling techniques. It provides low computation time and low accuracy.

Based on the related work presented above, the following drawbacks are observed by the previous methodologies used. Existing methods, such as Mehmood et al. [24], primarily focused on traditional machine learning techniques combined with feature selection. While these methods achieve reasonable accuracy, they often lack the ability to capture complex relationships within the data. In contrast, the proposed SDP-ESA-CycleGAN-POA method utilizes advanced machine learning techniques, specifically the ESA-CycleGAN model, which incorporates self-attention mechanisms. This allows the proposed approach to focus on relevant features while disregarding less significant ones, leading to improved accuracy and efficiency in defect prediction. The optimally trained convolutional neural network presented by Balasubramaniam and Gollagi [25] relies on a single type of architecture. The proposed method integrates the ESA-CycleGAN model with the POA, which enhances the model's learning capabilities and optimizes the weight parameters effectively.

3 Proposed methodology

In this section, the proposed SDP-ESA-CycleGAN-POA is discussed. The block diagram of the proposed software defect prediction using ESA-CycleGAN with pelican optimization system is shown in Fig. 1.

The process begins with the PROMISE dataset, which is pre-processed by parsing the source code into abstract syntax trees (AS-Trees), followed by handling class imbalance issues. The pre-processed data is then used for software defect prediction, leveraging edge features and self-attention-based cycle-consistent GANs to distinguish between "Buggy" and "Clean" code. After generating the defect predictions, the results are further optimized using the pelican optimization algorithm to enhance the accuracy and effectiveness of the defect prediction model.

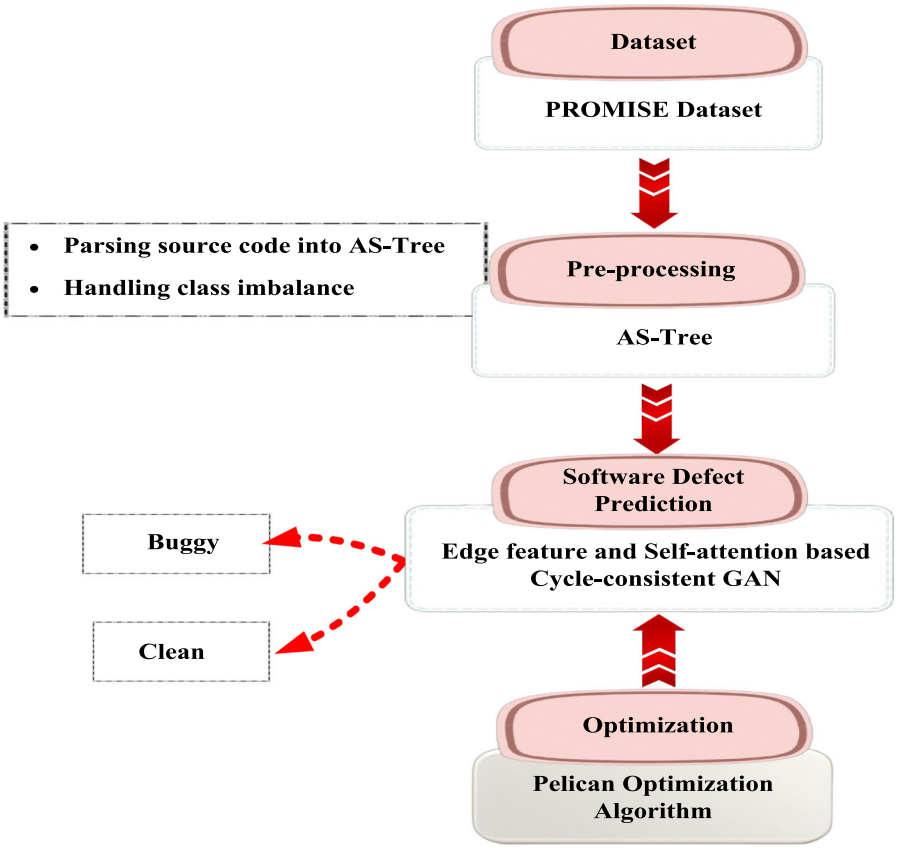

Fig. 1 Block diagram of proposed enhanced software defect prediction using ESA-CycleGAN with pelican optimization system

3.1 Data Pre-processing

The data pre-processing section involves two main tasks: parsing source code into abstract syntax tree (AS-Tree) tokens and managing class imbalance in the dataset.

3.1.1 Parsing source code into AS-tree

Java source code files from the PROMISE dataset are first parsed into abstract syntax trees (AS-Trees). It is a representation of the source code's syntactic structure in a tree. Each node in the AS-Tree corresponds to a particular construct in the code, such as control structures, method declarations or class instantiations. The purpose of using AS-Trees is to extract meaningful, structured features from the code that are essential for defect prediction. Key categories of AS-Tree nodes are tokenized, which are given below.

- *Control Flow Nodes:* This includes constructs like if, while, for and do statements, which dictate the flow of execution in the code. These nodes are recorded by their type (e.g. if statement, while loop) to capture the control flow patterns.
- *Method Calls and Class Instantiations:* Nodes representing method invocations and object creations are captured as tokens. For instance, the method call foo () is tokenized simply as foo without the parentheses, then focusing on the names of methods or classes without considering their arguments.
- *Declaration Nodes:* These include method and class declarations, which are tokenized by their respective names.

Once these key nodes are extracted, the less important nodes, such as comments, literals and other irrelevant code constructs are filtered out to focus on critical features to defect prediction. The remaining tokens are converted into integer vectors via the dictionary-based mapping, where each unique token is assigned a corresponding integer value.

3.1.2 Handling class imbalance

The datasets in SDP exhibit a significant class imbalance, here a number of defect-free instances greatly exceeding a number of defective ones. This imbalance can lead to biased model training, where the model becomes overly optimistic about predicting clean instances while neglecting the defective instances. Synthetic minority oversampling technique (SMOTE) effectively addresses this problem by generating synthetic instances of the minority class, thereby providing a more balanced dataset for training. Unlike simple oversampling techniques that duplicate existing instances of the minority class, SMOTE generates new synthetic instances by interpolating among the existing minority class examples. This approach increases a number of instances in the minority class and introduces variability and diversity in the training data. By generating new examples that are similar but not identical to existing ones, SMOTE helps the model learn more generalized patterns associated with defects, which can improve its predictive performance. SMOTE maintains the underlying distribution of the data better than random oversampling. The step-by-step procedure is deliberated how the SMOTE works:

***Step 1:* Identify minority class** It determines which class is the minority class (defective files in software defect prediction).

***Step 2:* Select a sample** Randomwise select the instance from the minority class.

***Step 3:* Find nearest neighbours** From the minority class, calculate the k-nearest neighbours ($k = 5$) of the selected instance. This involves measuring the distance between the selected instance and other instances in the minority class.

***Step 4:* Generate synthetic instances** Create a synthetic instance by interpolating amid the selected instance and neighbour.

***Step 5:* Repeat** Repeat the procedure until the desired count of synthetic instances is generated to balance the dataset.

By using SMOTE, the proposed model can learn better from the minority class and it improves its ability to predict instances of that class effectively.

3.2 Software defect prediction with edge feature and self-attention-based cycle-consistent generative adversarial network

This section discusses about the ESA-CycleGAN-POA [22] for software defect prediction. Here, the ESA-CycleGAN is employed to forecast a novel source file as buggy or clean. The integration of edge features permits the proposed technique to focus on critical structural aspects of software metrics, leading to more accurate defect classification. The self-attention mechanism further enhances this by enabling the model to prioritize and weigh the relevant parts of input data, which is essential for identifying subtle defect-related patterns. The self-attention mechanism's capability to capture the longer-range dependencies and interactions within the data further aids in understanding complex defect patterns, resulting in more informed classification decisions. Moreover, it can handle lengthy token sequences. The generator, edge extraction module (EEM), discriminator and self-attention module (SAM) are the four factors of the ESA-CycleGAN model.

3.2.1 Generator

The generator in the CycleGAN architecture for SDP is responsible for transforming input data, specifically the dense vector representations of AS-Tree tokens into outputs that indicate whether the software is buggy or clean. The dense vectors capture semantic relationships between tokens, also allowing the model to understand the context and meaning of the code constructs. After the AS-Tree tokens are embedded into dense vectors, they are organized into a suitable format for input into the GAN. This involves structuring the vectors into sequences that can be processed by the generator. Then the generator utilizes these dense vector representations to generate outputs that classify the input code as either buggy or clean. The integration of dense vector embeddings enhances the model's capacity to learn complex patterns and relationships within the code, ultimately improving the accurateness of software defect prediction. Generator consists of multiple processing layers, such as fully connected, convolutional layers, which learn to capture complex patterns along relationships within the input data. The generator's primary function is to produce realistic outputs that mimic the characteristics of actual software code, thereby generating a probability distribution that classifies the input as either buggy or clean. This output is crucial for the discriminator, which evaluates the generated data against real examples. The generator also incorporates a self-attention mechanism (SAM) to enhance its ability to focus on relevant features within the input, allowing it to produce outputs that are more aligned with the characteristics of buggy or clean code. Through this process, the generator plays a vital role in the adversarial training

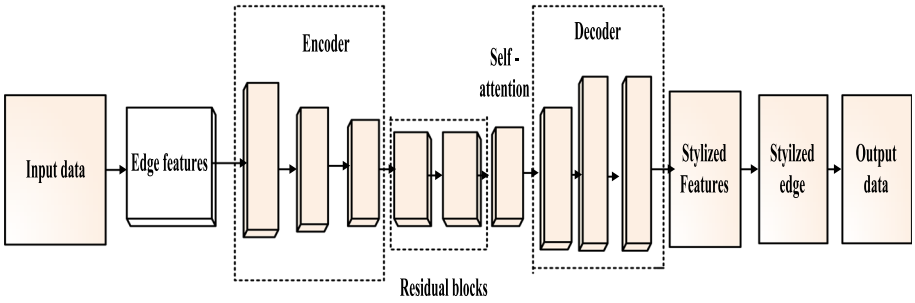

Fig. 2 Architecture diagram of ESA-CycleGAN generator

framework, continuously improving its outputs based on feedback from the discriminator until both components reach a state of Nash equilibrium.

Figure 2 shows the architecture diagram of ESA-CycleGAN generator. The inputs are processed through an encoder using residual blocks to capture the underlying defect patterns. A self-attention mechanism is applied between the encoder and the decoder to enhance the feature representation. The decoder then reconstructs the predicted defect features and patterns, which are output alongside the corresponding predicted defect status.

3.2.2 Discriminator

The discriminator in the CycleGAN architecture for SDP is tasked with distinguishing between real and generated data, specifically analysing the features extracted from both clean and buggy code. It receives inputs from the generator, which includes the outputs it has produced, and evaluates these alongside real data to classify them as either buggy or clean. The discriminator outputs a one-dimensional score that reflects the likelihood of the input belonging to a particular class, effectively serving as a critical component in the adversarial training process. By continuously training to maximize its classification accuracy, the discriminator provides essential feedback to the generator, pushing it to improve its outputs. This adversarial dynamic continues until both the generator and discriminator reach a state of Nash equilibrium, where neither can significantly improve without the other adapting, thus enhancing the overall presentation of the software defect prediction model.

Figure 3 shows the architecture diagram of ESA-CycleGAN discriminator diagram. The discriminator processes the output data, which could be defect predictions, through a series of self-attention layers. These layers refine the representation of the data, also enabling the model to distinguish between buggy and clean software based on the learned patterns. The self-attention layers output is used to classify the input as either "Buggy" or "Clean".

3.2.3 Edge extraction module (EEM)

The EEM is an important component of the CycleGAN architecture that focuses on identifying and emphasizing the structural elements within the software code. It operates by extracting high-frequency features, such as edges and textures on the input code, which are indicative of potential defects. By isolating these edge features, the EEM enhances the input representation that is fed into the generator, allowing the model to capture important structural information that may signal defects. This process improves the quality of the data being

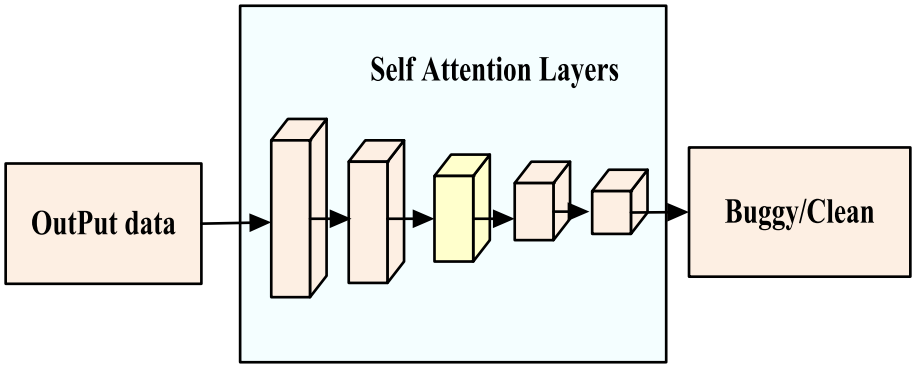

Fig. 3 Architecture diagram of ESA-CycleGAN discriminator

processed and also aids in the overall defect prediction accuracy by ensuring that the model is attuned to the critical aspects of the code's structure.

3.2.4 Self-attention mechanism (SAM)

The SAM is an integral part of the CycleGAN architecture that enhances the model's capacity to focus on related features within the input data. SAM assigns varying levels of importance to different parts of the input and allowing the model to prioritize features that are more indicative of defects while disregarding less relevant information. This mechanism is particularly beneficial in software defect prediction, where certain code constructs may carry more significance than others. Integrated into both the generator and discriminator, SAM improves the quality of the outputs generated by the generator and enhances the classification accuracy of the discriminator by ensuring that the most pertinent features are emphasized during the evaluation process. By capturing longer-range dependencies and contextual information, SAM enables the model to exploit the full range of feature information available in the input data, ultimately leading to more accurate defect predictions.

Adversarial loss: The adversarial loss measures how well the generator H is able to produce data that is indistinguishable from real data, as judged by the discriminator D. The style transfer technique is constructed in sample space of 2 domains to transmit among a and b domains. For mapping domain a to b utilizing CycleGAN method, H is utilized to signify this mapping, and H represents generator in the generative adversarial network for transforming the imagery in a domain of image in target b domain then evaluate regarding if the data is real using discriminator D. The generative adversarial loss is calculated by Eq. (1)

$$Loss_{GAN}(H, D, a, b) = E_{b`\sim P_{data}(b)}\left[\log D(b)\right] + E_{a\sim P_{data}(a)}\left[\log(1 - D(H(a)))\right] \quad (1)$$

where E_a represents the parameters of the generator for domain a, E_b indicates the parameters of the generator for domain b, $P_{data}(b)$ implies distribution of real data and $P_{data}(a)$ implies distribution of noise input to the generator. The final objective is given in Eq. (2),

$$H = \arg\min_{H}\max_{D} Loss_{GAN}(H, D, a, b) \quad (2)$$

This indicates that the generator aims to minimize the loss while the discriminator aims to maximize it. Finally, the proposed ESA-CycleGAN predicts the input file as buggy or clean. The artificial intelligence-based optimization strategy is used in the ESA-CycleGAN

classifier because of its convenience and pertinence. In this work, POA is utilized to improve the ESA-CycleGAN optimum parameter E_a and E_b. POA is employed to adjust the weight with bias parameter of ESA-CycleGAN.

3.3 Optimization utilizing pelican optimization algorithm (POA)

The weight parameters E_a and E_b of ESA-CycleGAN are optimized using POA [23] is discussed. Here, the parameters E_a is optimized to increase the accuracy,E_b is optimized to decrease computation time. POA excels at global optimization, making it suitable for finding optimal solutions in multidimensional search spaces. The algorithm demonstrates resilience and adaptability in dynamic and unpredictable environments. This characteristic allows POA to adjust to changes in the optimization landscape, enhancing its effectiveness in real-world applications. Also POA effectively balances exploration and exploitation for achieving high accuracy in defect prediction models. The proposed POA was inspired by the typical hunting behaviour of pelicans. Pelicans that look for food sources are called search agents in POA. POA offers several notable advantages in the realm of optimization. It excels at global optimization, ensuring it can find optimal solutions in complex and multidimensional search spaces. It exhibits resilience and adaptation, which makes it appropriate for dynamic and unpredictable surroundings. It takes inspiration from the foraging activity of pelicans. POA's parallel processing capability leverages modern computing resources for faster convergence. It maintains diversity within the population and preventing premature convergence to suboptimal solutions. The stepwise procedure of POA is delineated below,

3.3.1 *Step 1:* Initialization

The POA algorithm is depending upon population. All pelican members of this algorithm have possible solutions. The initialization of each member in the optimization process is done randomly using Eq. (3),

$$x_i = l_b + ran * (u_b - l_b)\, i = 1, 2, ..., M \tag{3}$$

where x_i denotes value of candidate solution, M signifies amount of population members, ran is a rand vector of 0, 1 interval, l_b implies b^{th} lower bound and u_b signifies b^{th} upper bound for problem variables. Figure 4 shows the flow chart of POA.

3.3.2 Step 2: Random generation

Create the input parameters randomwise. The optimum fitness selection depends upon explicit hyper parameter situation.

3.3.3 *Step 3:* Fitness function estimation

Fitness function is estimated with optimization parameter value for increasing weight parameter E_a and E_b of ESA-CycleGAN. It is given in Eq. (4),

$$Fitness\ function = optimzing[E_a\ \ and\ \ E_b] \tag{4}$$

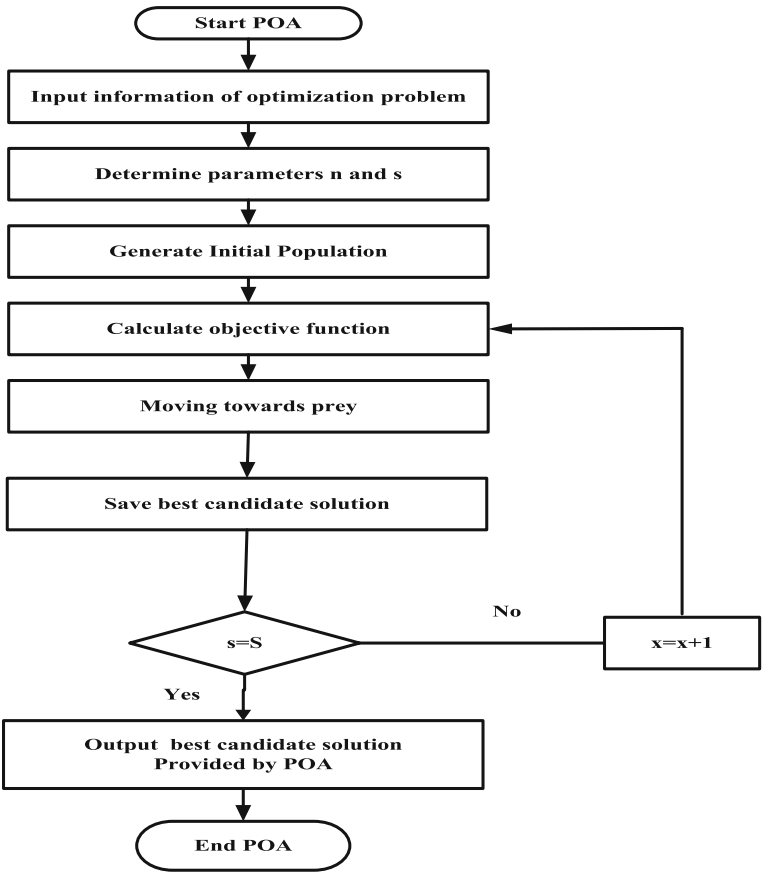

Fig. 4 POA flow chart

3.3.4 *Step 4:* Exploration phase

In this phase, pelicans search for new positions based on their current locations and the positions of the best-performing pelicans. For example, first pelican may randomly adjust its position towards the best-performing pelican while exploring new areas. This pelican's technique allows for search space scanning along exploration capabilities of POA in identifying diverse search space locations. The idea that the prey's location is randomly generated in the search space is essential to POA. As a result, POA may precisely search the problem-solving domain with greater exploration capacity. Equation (5) expresses the previously mentioned ideas as well as the pelican's route for attaining the prey position.

$$x_i^{n-1} = \begin{cases} x_i(a) + ran.(x_r - T.x_i), & G_b < G_i \\ (x_i)x_i(a) - ran.(x_r - T.x_i), else \end{cases} \tag{5}$$

where x_i^{n-1} denotes novel i^{th} pelican status in j^{th} dimension depending upon phase 1, T implies random number equivalent to 1 or 2, x_r signifies prey location in j^{th} dimension, G_b implies objective function. If the goal function value is higher in the new pelican location,

then it is accepted. This kind of updating referred to as effective updating, keeps the algorithm from wandering into less-than-ideal regions. This procedure is expressed in Eq. (6),

$$X_i = \begin{cases} X_i^{x_1} & G_i^{x_1} < G_i \\ X_i, else \end{cases} \tag{6}$$

where $X_i^{x_1}$ denotes novel status of i^{th} pelican and $G_i^{x_1}$ signifies objective function value depend upon phase.

3.3.5 *Step 5:* Exploitation phase

Pelicans refine their positions based on the best-known positions in the population. For instance, if one pelican has the best position, other pelicans may move closer to it. It support enhancing local search's potential. Equation (7) represents the mathematical model for the phase 2 solution,

$$x_i^{n-2} = x_i(a) + P.(1 - a/N).(2.ran - 1)x_i(a), \tag{7}$$

where N implies maximum quantity of iterations, a implies current iteration and P implies constant equal to 0.2. The new solution based on novel position is expressed as Eq. (8),

$$X_i = \begin{cases} X_i^{x_2} & G_i^{x_2} < G_i \\ X_i, else \end{cases} \tag{8}$$

where $X_i^{x_2}$ implies prospective solution and $G_i^{x_2}$ refers objective function value.

3.3.6 *Step 6:* Termination

Each iteration allows the pelicans to improve their positions based on the feedback from the objective function. The weight parameter of E_a and E_b from the ESA-CycleGAN are optimized using POA, will iteratively repeat step 3 until fulfil the halting criteria $x = x + 1$. The best position found during the search is considered the optimal solution. Finally, the SDP-ESA-CycleGAN-POA method effectively predicts software defects with high accuracy and low error rate.

3.4 Computational complexity

This subsection computes the computational complexity of SDP-ESA-CycleGAN-POA. Here, $O(N)$ denotes computational complexity of algorithm initialization processes. Every member of populace assesses the objective function in all iteration of both stages. Thus, $O(2 \cdot T \cdot N)$ denotes computational complexity for the estimation of fitness function. The provided prey is produced and assessed in every iteration. Let $O(T) + O(T \cdot m)$ represents computational complexity for prey generation, N populace members including m dimensions are updating in 2 phases. Here, computational complexity for solutions updation is denoted as $O(2 \cdot T \cdot N \cdot m)$. The SDP-ESA-CycleGAN-POA whole computational complexity is equivalent to $O(N + T(1 + m) \cdot (1 + 2 \cdot N))$.

4 Results and discussion

The simulation outcomes of the SDP-ESA-CycleGAN-POA are discussed in this section. The experiments are implemented in Windows laptop using Intel(R) Core(TM) i5-2410 M processor, 6 GB of primary storage with 1 TB of secondary storage. The proposed method is implemented in Python. The Python packages pandas, numpy, seaborn, matplotlib, sklearn and pyswarm are used for constructing the machine learning predictor. The key algorithm parameters used in the proposed model includes 100 population size, with maximum count of iterations set to 100 and 30 runs per function. The population size as well as iterations count strike a balance between thorough exploration and computational efficiency. The learning rate is set at 0.01, which influences how the model adjusts its weights. Additionally, a dropout rate of 0.5 is applied, indicating that 50% of neurons are dropped during training to prevent overfitting. The proposed SDP-ESA-CycleGAN-POA method is compared with other SDP-RF-LR [24], SDP-OT-CNN [25], SDP-RF-SVM-ANN [26] and SDP-RBFNN-FS [27] models, these are chosen for their diversity in methodologies, relevance and prominence in the field. These models cover a broad range of approaches, from ensemble learning and probabilistic models (SDP-RF-LR), to deep learning (SDP-OT-CNN) and kernel-based methods (SDP-RBFNN-FS), providing a comprehensive backdrop for evaluating the proposed method. The selected models are well established in the literature and have been widely used in previous studies for software defect prediction. By comparing the proposed method against these benchmarks, the paper provides a clear context for evaluating its performance.

4.1 Datasets

In this work, the CV-P-SC (cross-version PROMISE source code) and CP-P-SC (cross-project PROMISE source code) [31] datasets are selected for evaluating SWDP models due to their suitability for SDP task. The datasets are sourced from the PROMISE repository is a widely recognized platform for sharing datasets related to software engineering research.

CP-P-SC (Cross-project PROMISE source code): This dataset contains source code and bug labels for multiple versions of each project. It is specifically designed for training and evaluating SWDP models that can predict the probability of a software file being defective in a different project. This cross-project prediction scenario is essential for assessing the generalizability and transferability of SDP models across different software projects. By testing models on projects, they have not been trained to evaluate performance of the models for adapting adapt to new project contexts and identify defects effectively.

CV-P-SC (Cross-version PROMISE source code): Similar to the CP-P-SC dataset, the CV-P-SC dataset includes source code and bug labels for multiple versions of each project. However, it is tailored for cross-version SDP, where models are trained and evaluated to predict the probability of a software file being defective in a novel version of the same project. Cross-version prediction is essential for assessing the robustness and consistency of SDP models over different iterations of the same software project. It helps in understanding how well the models can maintain predictive performance as the software evolves and new versions are released.

By utilizing both the CP-P-SC and CV-P-SC datasets, it is possible to comprehensively evaluate SDP models in diverse scenarios, covering both cross-project and cross-version prediction challenges. The PROMISE database is typically separated as 80% for training and 20% for testing to assure a feasible evaluation of the software defect prediction models. This

division allows the methods to learn from a substantial portion of the data while reserving a significant subset for testing their predictive capabilities on unseen instances.

4.2 Performance metrics

The proficiency of the proposed approach is examined with the mentioned metrics. The following confusion matrix is essential to assess these metrics.

- True Positive (TP): Count of predicted defective software modules
- True Negative (TN): Count of properly predicted clean software modules.
- False Positive (FP): Count of clean software inaccurately forecasted as defective.
- False Negative (FN): Count of defective software inaccurately forecasted as clean.

4.2.1 Accuracy

This computes the proportion of accurately predicted samples to total samples in the dataset. This is calculated using Eq. (9),

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \tag{9}$$

4.2.2 Precision

Out of all the positive predictions made, this calculates the rate of true positive predictions by Eq. (10),

$$\text{Pr}ecision = \frac{TP}{(TP + FP)} \tag{10}$$

4.2.3 Sensitivity

It calculates the ratio of true positives that accurately recognized by the method using Eq. (11),

$$Sensitivity = \frac{TP}{TP + TN} \tag{11}$$

4.2.4 Specificity

This computes the ratio of appropriately identified actual negatives using the model. It is measured through Eq. (12),

$$Specificity = \frac{TN}{TN + FP} \tag{12}$$

4.2.5 F-Measure

This is the harmonic mean of precision as well as sensitivity. It is computed using Eq. (13),

$$F - measure = 2 \times \frac{recall \times precision}{recall + precision} \tag{13}$$

4.2.6 Matthew's correlation coefficient (MCC)

This is the amount of classifications quality and is determined using Eq. (14),

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{14}$$

4.3 Simulation analysis

Figures 5, 6, 7, 8, 9, 10, 11, 12 show the simulation result of SDP-ESA-CycleGAN-POA method. The performance metrics are analysed with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS methods.

Figure 5 depicts accuracy analysis. The proposed model's edge characteristics and self-attention processes enable it to recognize and convey complex relationships and patterns in the data on software defects. The self-attention permits the proposed technique to focus on relevant sections of input, improving feature representation, edge features draw attention to structural elements. POA makes certain that the proposed approach attains more accuracy in defect prediction tasks by adjusting biases and weights in response to optimization findings. The proposed SDP-ESA-CycleGAN-POA method achieves 21.57%, 23.41%, 16.10%, 18.73% greater accuracy for buggy software and 17.43%, 35.87%, 19.59% and 22.96% higher accuracy for clean software when compare with the existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 6 determines precision analysis. Self-attention processes enable the proposed approach to balance the significance of diverse input data points when it is making predictions. This lets the proposed model concentrate on relevant features and filter out noise or unimportant data, which is very helpful for defect prediction. The proposed model's precision

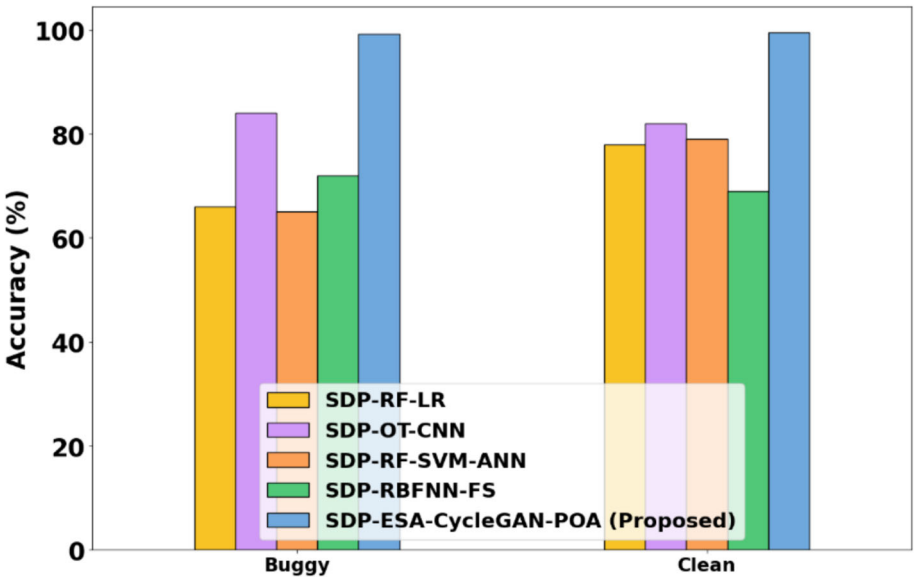

Fig. 5 Accuracy analysis

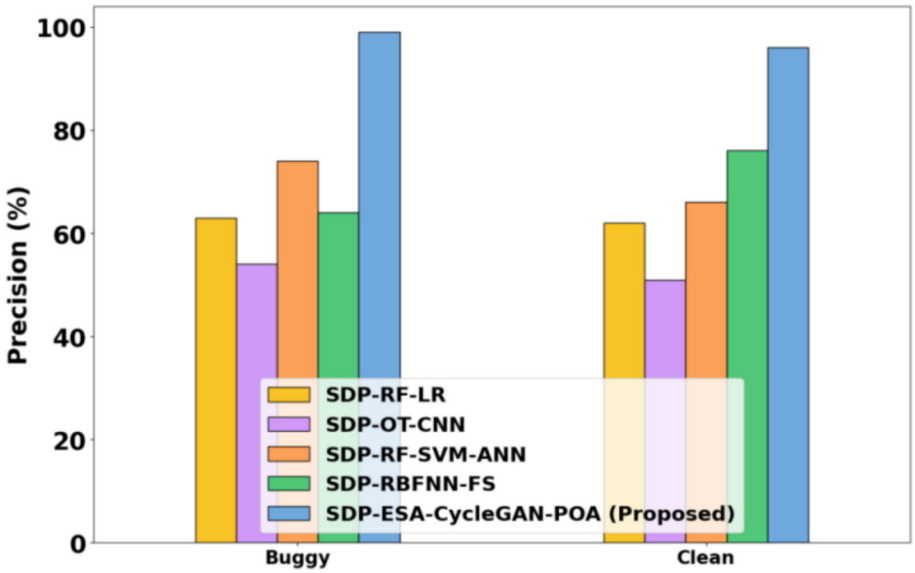

Fig. 6 Precision analysis

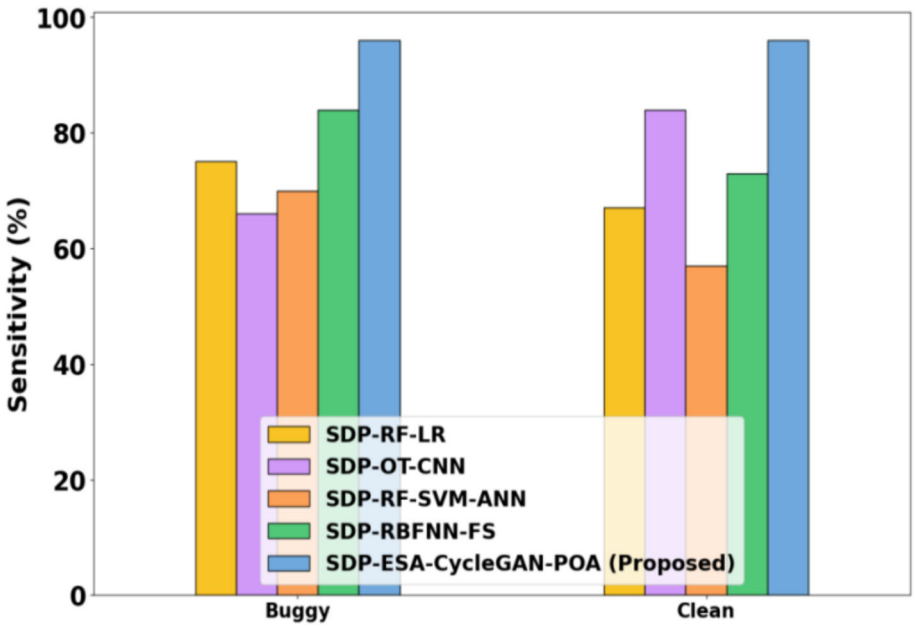

Fig. 7 Sensitivity analysis

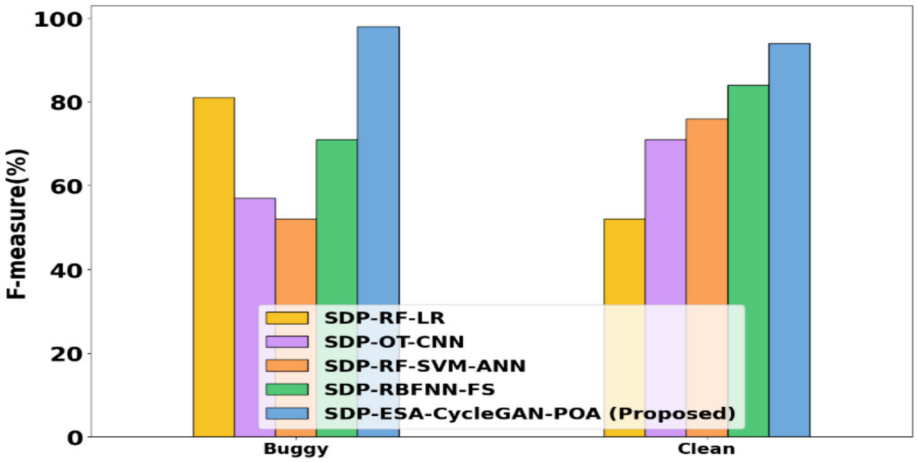

Fig. 8 F-measure analysis

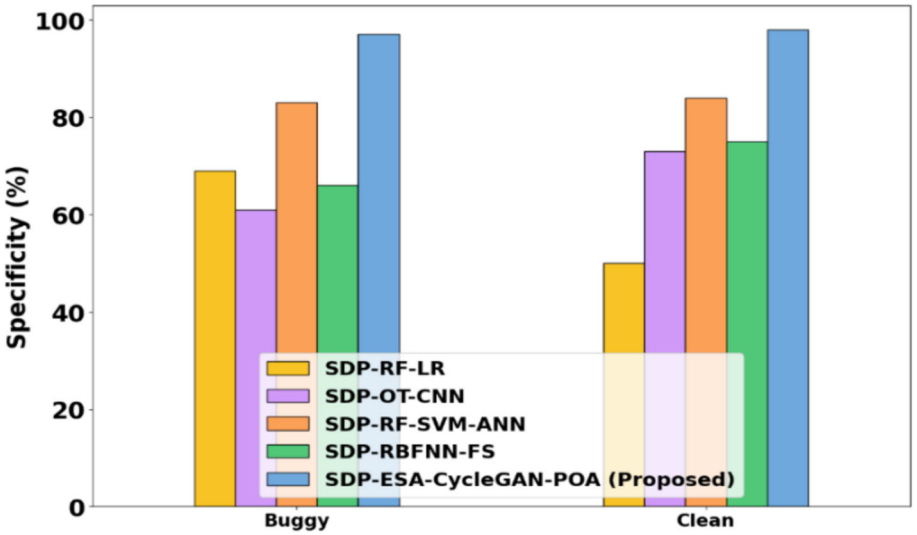

Fig. 9 Specificity analysis

increases as it pay attention to important features, which leads to more precise predictions. The proposed SDP-ESA-CycleGAN-POA method achieves 23.81%, 20.93%, 18.17% and 22.91% higher precision for buggy software and 18.27%, 30.04%, 26.37% and 17.09% higher precision for clean software when compare with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 7 shows sensitivity analysis. The cycle consistency loss in the proposed model guarantees a consistent and reversible transition between software states that are buggy and clean. This transformational stability helps in preserving accuracy in defect identification while minimizing false positives. POA optimizes the parameters of proposed

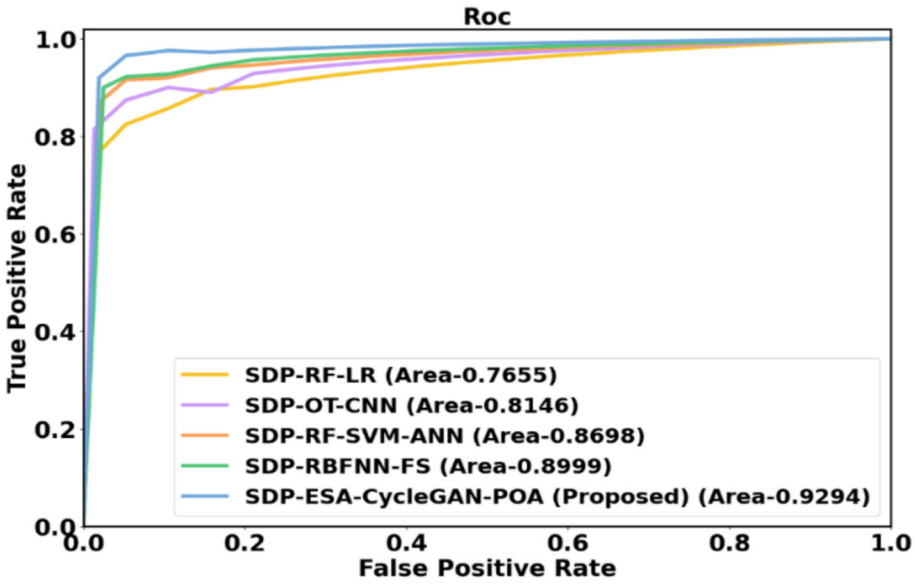

Fig. 10 RoC analysis

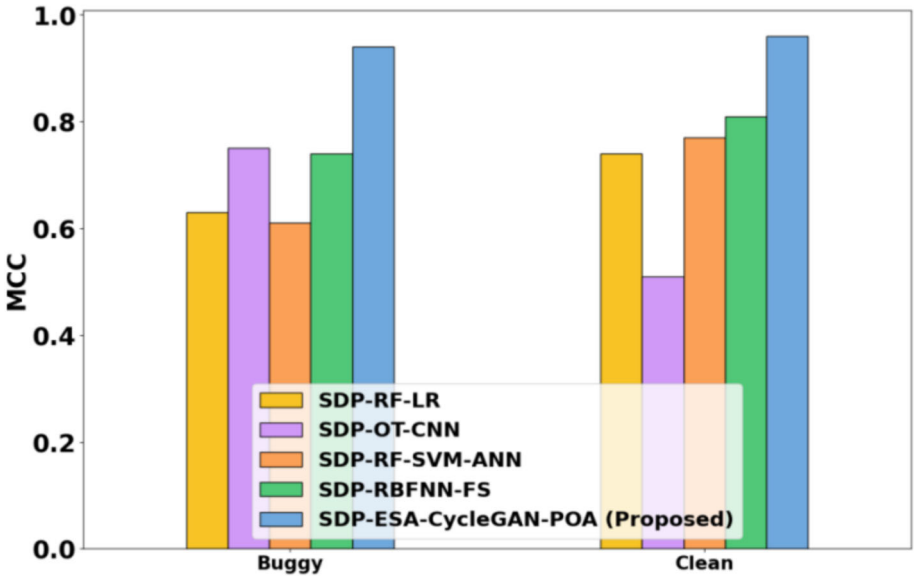

Fig. 11 MCC analysis

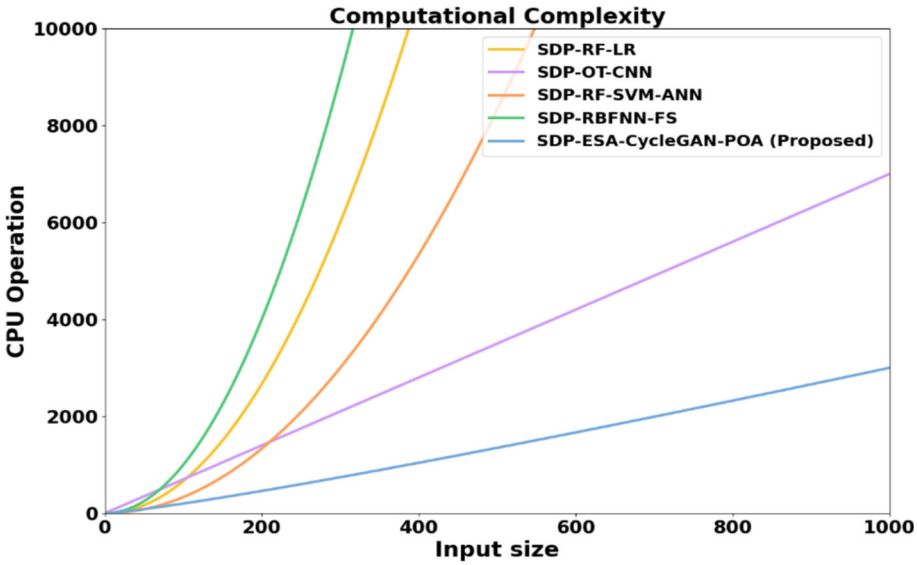

Fig. 12 Computational complexity analysis

SDP-ESA-CycleGAN-POA model, potentially improving its ability to detect defects while maintaining high sensitivity. The proposed SDP-ESA-CycleGAN-POA method achieves 19.34%, 22.91%, 20.26% and 29.28% higher sensitivity for buggy software and 25.20%, 17.98%, 20.39% and 27.62% higher sensitivity for clean software compared with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 8 depicts F-measure analysis. In the proposed model, incorporating ESA-CycleGAN principles improves the model's capability to learn and transfer features across different distributions or domains of software data. POA optimizes the parameters of proposed ESA-CycleGAN model effectively. It adjusts the proposed model's weights and biases to minimize prediction errors and maximize accuracy in distinguishing between buggy and clean software instances. The proposed SDP-ESA-CycleGAN-POA method achieves 23.17%, 19.95%, 28.32% and 16.58% higher F-measure for buggy software and 26.46%, 20.37%, 16.06% and 23.50% higher F-measure for clean software compared with existing such as SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 9 shows specificity analysis. Here, POA balances exploration and exploitation during optimization, which prevents the proposed technique from getting stuck in local minima and ensures a global optimum for better defect prediction, which improves the specificity. The proposed SDP-ESA-CycleGAN-POA method achieves 17.97%, 22.26%, 23.96% and 19.52% higher specificity for buggy software and 23.96%, 24.49%, 18.53% and 31.87% higher specificity for clean software when compare with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS techniques.

Figure 10 shows RoC analysis. The high RoC value for the proposed model is due to effective edge feature extraction capturing key structural information, the self-attention mechanism highlighting relevant features, CycleGAN generating high-quality synthetic data and the pelican optimization algorithm efficiently tuning parameters for optimal performance. The proposed SDP-ESA-CycleGAN-POA method achieves 20.84%, 25.98%, 16.70% and

Table 1 Computational time and memory usage analysis

Methods	Computation time (sec)	Memory usage (MB)
SDP-RF-LR	349.4	524
SDP-OT-CNN	259.1	510
SDP-RF-SVM-ANN	386.5	472
SDP-RBFNN-FS	196	495
SDP-ESA-CycleGAN-POA (proposed)	93.86	438

30.45% higher RoC when compare with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 11 shows the MCC analysis. The cycle-consistent nature of CycleGANs ensures that the transformations between buggy and clean software are consistent and understandable. This transparency helps in validating the proposed model's predictions. The self-attention mechanism in the ESA-CycleGAN enables the technique to assign importance to different parts of the input, making it easier to interpret which features contribute most significantly to defect prediction. The proposed SDP-ESA-CycleGAN-POA method achieves 19.42%, 20.67%, 24.92% and 19.52% higher MCC for buggy software and 26.76%, 20.92%, 30.92% and 23.54% higher MCC for clean software when compare with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Table 1 shows computation time and memory usage analysis. The hyperparameters of ESA-CycleGAN are optimized using POA, which fine-tunes the network to function effectively by reducing unnecessary computations and speeding up the training process. The combination of edge features and self-attention mechanisms leads to a network architecture that is both compact and powerful. This result in reduced computational time and memory usage of the proposed model compared to more complex models. The proposed SDP-ESA-CycleGAN-POA method achieves 16.90%, 21.92%, 19.06% and 27.51% lower computational time and 20.04%, 17.47%, 23.28% and 29.24% lower memory usage when compared with existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models.

Figure 12 depicts computational complexity analysis. The analysis indicates that the SDP-ESA-CycleGAN-POA method demonstrates a linear improvement in CPU operation time and memory usage. This indicates that the computational resources used by the proposed model scale more effectively than those of the current models as the size of the input data increases. The reduced computational complexity implies that the SDP-ESA-CycleGAN-POA model is effective in terms of accuracy and precision and manageable in terms of resource consumption. The linear improvement in computational complexity displays that the SDP-ESA-CycleGAN-POA method is scalable. This analysis shows the proposed model can be trained more quickly and deployed more efficiently.

Table 2 tabulates the benchmark table with literature support. The simulation results demonstrate from Table 1 that the SDP-ESA-CycleGAN-POA achieves 21.57%, 23.41%, 16.10%, 18.73%,24.39%, 24.36% and 19.13% higher accuracy with existing models, like Mehmood, et al. [24], Balasubramaniam [25], Ali, et al. [26], Alkhasawneh [27], Rajnish, et al. [28], Parashar, et al. [29] and Manchala [30] models. The SDP-ESA-CycleGAN-POA achieves 23.81%, 20.93%, 18.17%, 22.91%, 12.37%, 30.78% and 24.16% greater precision with existing methods, like Mehmood, et al., [24], Balasubramaniam, [25], Ali, et al., [26],

Table 2 Some benchmark table with literature support

Performance Analysis	Authors							
	Mehmood, et al., [24]	Balasubramaniam., [25]	Ali, et al., [26]	Alkhasawneh, [27]	Rajnish, et al., [28]	Parashar, et al., [29]	Manchala, [30]	SDP-ESA-CycleGAN-POA(proposed)
Accuracy (%)	66.15	84.52	65.32	72.61	79.01	85.56	74.84	99.28
Precision (%)	63.39	54.15	74.16	64.92	83.15	81.36	83.32	99.13
Specificity (%)	69.91	61.33	83.60	66.74	79.95	79.10	79.90	98.90
Sensitivity (%)	75.50	66.64	70.15	84.04	69.93	77.96	75.97	98.86
F-measure (%)	81.36	57.21	52.10	71.17	79.20	84.16	81.36	98.07
RoC	0.76	0.81	0.86	0.89	0.88	80.36	0.73	0.99
Computation time (s)	349.4	259.1	386.5	196.0	261.8	215.9	198.3	93.86

Table 3 Comparative analysis using Bug prediction dataset

Methods	Accuracy (%)	Precision (%)	Specificity (%)	Computational time (sec)
SDP-RF-LR	67.62	64.41	68.04	355.04
SDP-OT-CNN	83.41	56.27	60.97	362.57
SDP-RF-SVM-ANN	64.38	72.63	82.91	390.24
SDP-RBFNN-FS	73.67	65.37	67.41	384.64
SDP-ESA-CycleGAN-POA (proposed)	99.23	99.10	98.74	98.36

Alkhasawneh, [27], Rajnish, et.al. [28], Parashar, et.al. [29] and Manchala [30] respectively. The SDP-ESA-CycleGAN-POA attains 17.97%, 22.26%, 23.96%, 19.52%, 23.16%, 26.97% and 21.30% greater specificity with existing models, like Mehmood, et al., [24], Balasubramaniam [25], Ali, et al., [26], Alkhasawneh, [27], Rajnish, et al., [28], Parashar, et al., [29] and Manchala, [30] methods.

4.4 Comparative analysis using bug prediction dataset

This segment offers a comprehensive comparative analysis of various machine learning methods applied to SDP using a bug prediction dataset [32]. The evaluation focuses on the metrics, like accuracy, precision, specificity and computational time.

Table 3 displays the comparative analysis using Bug prediction dataset. The SDP-ESA-CycleGAN-POA achieves the highest performance for accuracy of 99.23%, precision of 99.10% and specificity of 98.74%, while also having the lowest computational time of 98.36 s. Other methods, such as SDP-RF-LR and SDP-OT-CNN, exhibit lower accuracy and precision, and relatively higher computational times, indicating that the proposed model offers both superior prediction performance and efficiency in comparison to the other models.

4.5 Statistical test

In the statistical analysis, the Wilcoxon signed-rank along Cliff's δ test is utilized for displaying the importance of performance difference among the proposed and other techniques. The nonparametric Wilcoxon signed-rank test is a statistical hypothesis test. The proposed study uses ρ-value to validate whether performance difference of the model is statistically significant at 0.05 significance level. Cliff's δ is used in the experiment to determine the amount of variance in the models' prediction presentation. The dimension of variance is separated as 4 levels. In summary, if ρ-value is lesser than 0.05 as well as Cliff's δ is higher or equivalent to 0.145 this is an important variance in performance estimation of techniques. Table 4 tabulates the Cliff's δ values along its effective levels.

Table 5 shows the comparison outcomes of Wilcoxon signed-rank test and Cliff's delta. The proposed SDP-ESA-CycleGAN-POA has the smallest p-value (0.023) and the highest effect size (0.34), suggesting it performs better compared to the other models, which have lower effect sizes, such as 0.21 for SDP-RF-LR and 0.12 for SDP-RF-SVM-ANN.

ANOVA (analysis of variance) is a statistical method determines whether there are any significant variations among the means of three or more groups. It helps to test if the observed

Table 4 Cliff's δ along its effective levels

Cliff's δ	Effective levels
$\|\delta\| < 0.145$	Negligible
$0.145 \leq \|\delta\| < 0.33$	Small
$0.33 \leq \|\delta\| < 0.472$	Medium
$0.472 \leq \|\delta\|$	Large

Table 5 Comparison outcomes of Wilcoxon signed-rank test and Cliff's delta

Model	Wilcoxon signed-rank test (p-value)	Cliff's delta (Effect Size)
SDP-RF-LR	0.045	0.21
SDP-OT-CNN	0.078	0.18
SDP-RF-SVM-ANN	0.091	0.12
SDP-RBFNN-FS	0.087	0.15
SDP-ESA-CycleGAN-POA (proposed)	0.023	0.34

Table 6 ANOVA

Source of variation	Sum of Squares (SS)	Degrees of freedom (df)	Mean square (MS)	F-statistic	p-value
Between groups	30.5	2	15.25	5.67	0.012
Within groups	54.0	27	2.00		
Total	84.5	29			

variation in a dataset can be attributed to the independent variable being tested or if it is due to random chance.

Table 6 shows the ANOVA. The F-statistic of 5.67 specifies how much the between-group variance is compared to the within-group variance, and the p-value of 0.012 suggests that the differences between groups are statistically significant at a 5% significance level. This implies that the factors being tested, such as different software models, have a significant effect on the defect prediction performance.

4.6 Discussion

This paper highlights the significance of software defect prediction in identifying defects that can cause serious issues during software usage. Traditionally, defect prediction methods have primarily focused on the structural aspects of code, often supervising the semantic features inherent in software. Software defect prediction using ESA-CycleGAN with POA seeks to bridge this gap by using advanced techniques that can capture the subtleties of

code semantics. ESA-CycleGAN with its ability to analyse abstract syntax tree (AS-Tree) tokens and extract essential features provide a promising avenue for enhancing SDP accuracy. The incorporation of ESA-CycleGAN principles allows the model to effectively learn and transfer features across different distributions of software data. This capability enhances the proposed methods capacity to generalize from training data to unseen cases. The use of the POA optimizes the method's parameters, adjusting weights including biases to diminish prediction errors. This fine-tuning process is essential for improving the proposed model's accuracy in distinguishing between buggy and clean software instances.

The experimental outcomes on the open-source PROMISE dataset determines the superiority of the SDP-ESA-CycleGAN-POA method, with substantial improvements in accuracy, sensitivity, specificity and computational efficiency compared to the existing methods, like SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS. The SDP-ESA-CycleGAN-POA technique shows significant improvements in sensitivity for buggy software detection (up to 47.87% higher) and clean software detection (up to 56.46% higher) evaluated with existing techniques. In F-measure, the SDP-ESA-CycleGAN-POA technique achieves notable improvements for both buggy and clean software detection evaluated with SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS methods. This enhanced accuracy in defect prediction can help software developers identify problematic areas within the code base more efficiently and accurately. These outcomes indicate that the proposed SDP-ESA-CycleGAN-POA technique is effective in improving the performance of SDP techniques.

While the proposed method shows promising results it also has several limitations. The performance of the SDP-ESA-CycleGAN-POA model may differ when it is used with different datasets or in other industrial settings. As the complexity of software systems increases, understanding the proposed model's decision-making process becomes essential for trust and adoption in industry settings.

5 Threats to validity

The results of this investigation are impacted by several dangers. The threats to validity internal and external are discussed. The choice of measures is utilized as predictors associated with an internal threat. There are numerous metrics defined in the literature. Though the proposed method used the metrics that are available from the chosen datasets, other metrics are the better indicator of faults.

One of the primary concerns in any predictive modelling study is the ability to generalize results across different datasets. The proposed model's performance is evaluated using PROMISE database. However, this dataset may not capture the full diversity of software projects, coding practices and defect types encountered in real-world applications. As a result, the model's effectiveness may vary when applied to different datasets or in different contexts. For instance, software developed in different programming languages or frameworks may exhibit distinct defect patterns that the model has not been trained on, leading to reduced predictive accuracy. The robustness of the SDP-ESA-CycleGAN-POA model in large-scale, real-world applications is another critical consideration. While the model may perform well on the datasets used for training and testing, its performance in a production environment with large volumes of data and varying software architectures may differ significantly. The data quality, noise and dynamic nature of software development can introduce challenges

that are not present in the controlled experimental settings. The model's ability to adapt to these changes and maintain high performance is essential for its practical applicability.

The development of the proposed model involves complex algorithms and coding practices. Despite efforts to minimize coding errors, there is always a risk of bugs or inaccuracies in the implementation that could affect the performance of the method. Rigorous testing and validation against benchmark datasets are essential to assure the dependability of the method. The data analysis process itself can introduce external threats to validity. If the model is trained on biased or low-quality data, the predictions may not be reliable. Additionally, the method's performance is influenced by the specific features selected for analysis.

6 Conclusion

In this manuscript, the proposed SDP-ESA-CycleGAN-POA was successfully implemented. The manuscript proposes SDP-ESA-CycleGAN-POA as a novel approach for software defect prediction, leveraging advanced techniques such as ESA-CycleGAN and POA to enhance accuracy and efficiency. The performance metrics demonstrated the robustness and superiority of SDP-ESA-CycleGAN-POA over the existing models. The proposed SDP-ESA-CycleGAN-POA method attains 17.97%, 22.26%, 23.96% and 19.52% higher specificity when compared with the existing SDP-RF-LR, SDP-OT-CNN, SDP-RF-SVM-ANN and SDP-RBFNN-FS models. Future work could focus on adapting the proposed model to handle larger and more complex datasets. Conducting case studies in real-world software development environments would provide valuable insights into the practical applicability of the model and also explore methods such as attention mechanisms and feature visualization to enhance the interpretability of the proposed model.

Author contribution Dr. S.V. Gayetri Devi -(Corresponding Author)—Conceptualization Methodology, Original draft preparation Dr. P.V.V. Satyanarayana–Supervision Mr. A. Mani–Supervision Dr. S. Praveena—Supervision.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Meng F, Cheng W, Wang J (2021). Semi-supervised software defect prediction model based on tri-training. KSII Transactions on Internet & Information Systems. 15(11).
2. Wang H, Zhuang W, Zhang X (2021) Software defect prediction based on gated hierarchical LSTMs. IEEE Trans Reliab 70(2):711–727
3. Eken B, Tosun A (2021) Investigating the performance of personalized models for software defect prediction. J Syst Softw 181:111038
4. Feng S, Keung J, Yu X, Xiao Y, Zhang M (2021) Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Inf Softw Technol 139:106662
5. Uddin MN, Li B, Ali Z, Kefalas P, Khan I, Zada I (2022) Software defect prediction employing BiLSTM and BERT-based semantic feature. Soft Comput 26(16):7877–7891
6. Chen LQ, Wang C, Song SL (2022) Software defect prediction based on nested-stacking and heterogeneous feature selection. Complex Intell Syst 8(4):3333–3348
7. Giray G, Bennin KE, Köksal Ö, Babur Ö, Tekinerdogan B (2023) On the use of deep learning in software defect prediction. J Syst Softw 195:111537

8. Yuan Y, Li C, Yang J (2023) An improved confounding effect model for software defect prediction. Appl Sci 13(6):3459
9. Aftab S, Abbas S, Ghazal TM, Ahmad M, Hamadi HA, Yeun CY, Khan MA (2023) A cloud-based software defect prediction system using data and decision-level machine learning fusion. Mathematics 11(3):632
10. Ye T, Li W, Zhang J, Cui Z (2023) A novel multi-objective immune optimization algorithm for under sampling software defect prediction problem. Concurren Computat Pract Exper 35(4):e7525
11. Tang Y, Dai Q, Yang M, Du T, Chen L (2023) Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. Int J Mach Learn Cybern 14(6):1967–1987
12. Borandag E (2023) Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques. Appl Sci 13(3):1639
13. Liu C, Sanober S, Zamani AS, Parvathy LR, Neware R, Rahmani AW (2022) Defect prediction technology in software engineering based on convolutional neural network. Secur Commun Networks. https://doi.org/10.1155/2022/5058461
14. Liu Y, Xu C, Chen L, Yan M, Zhao W, Guan Z (2024) TABLE: Time-aware Balanced Multi-view Learning for stock ranking. Knowl Based Syst 303:112424
15. Zhu Q (2020) On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset. Pattern Recognit Lett 136:71–80
16. Pandey SK, Mishra RB, Tripathi AK (2021) Machine learning based methods for software fault prediction: a survey. Expert Syst Appl 172:114595
17. Thota MK, Shajin FH, Rajesh P (2020) Survey on software defect prediction techniques. Int J Appl Sci Eng 17(4):331–344
18. Nevendra M, Singh (2022) Empirical investigation of hyperparameter optimization for software defect count prediction. Expert Syst Appl 191:116217
19. Nevendra M, Singh P (2021) Software defect prediction using deep learning. Acta Polytech Hung 18(10):173–189
20. Nevendra M, Singh P (2021) Defect count prediction via metric-based convolutional neural network. Neural Comput Appl 33(22):15319–15344
21. Nevendra M, Singh P (2022) A survey of software defect prediction based on deep learning. Arch Comput Methods Eng 29(7):5723–5748
22. Wang L, Wang L, Chen S (2022) ESA-CycleGAN: edge feature and self-attention based cycle-consistent generative adversarial network for style transfer. IET Image Process 16(1):176–190
23. Trojovský P, Dehghani M (2022) Pelican optimization algorithm: a novel nature-inspired algorithm for engineering applications. Sensors 22(3):855
24. Mehmood I, Shahid S, Hussain H, Khan I, Ahmad S, Rahman S, Ullah N, Huda S (2023) A novel approach to improve software defect prediction accuracy using machine learning. IEEE Access 11:63579–63597
25. Balasubramaniam S, Gollagi SG (2022) Software defect prediction via optimal trained convolutional neural network. Adv Eng Softw 169:103138
26. Ali M, Mazhar T, Arif Y, Al-Otaibi S, Ghadi YY, Shahzad T, Khan MA, Hamam H (2024) Software defect prediction using an intelligent ensemble-based model. IEEE Access. https://doi.org/10.1109/ACCESS.2024.3358201
27. Alkhasawneh MS (2022) Software defect prediction through neural network and feature selections. Appl Comput Intell Soft Comput 2022(1):2581832
28. Rajnish K, Bhattacharjee V, Gupta M (2022) A novel convolutional neural network model to predict software defects. Fundamentals and Methods of Machine and Deep Learning: Algorithms, Tools and Applications. 211–35.
29. Parashar A, Kumar Goyal R, Kaushal S, Kumar Sahana S (2022) Machine learning approach for software defect prediction using multi-core parallel computing. Autom Softw Eng 29(2):44
30. Manchala P, Bisi M (2022) Diversity based imbalance learning approach for software fault prediction using machine learning models. Appl Soft Comput 124:109069
31. https://github.com/ApoorvaKrisna/NASA-promise-dataset-repository?tab=readme-ov-file
32. https://bug.inf.usi.ch/index.php

S. V. Gayetri Devi (Gayetri Devi Subramanian Venkhataraman) is currently working as Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College Of Engineering, Chennai, Tamil Nadu, India. Her main areas of work and interests include Software testing and Automation and Software Quality Assurance of Real Time Software.

P. V. V. Satyanarayana is currently working as Professor in the Department of Electrical and Electronics Engineering of Malla Reddy Engineering College, Hyderabad, India. His special fields of interest include Power Systems, Power Quality, Distributed Generation, Power Electronics, Non-conventional energy sources, HVDC, and FACTS.

A. Mani is currently pursuing Ph.D. from Anna University, Chennai, India. His research interests are in Cloud/Edge Computing, IoT, and Healthcare.

S. Praveena is an Associate professor, Department of Electronics and Communication Engineering, Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad-75, Telangana, India. Her research areas include Image Processing, Signal Processing, Deep Learning, and Computer Vision.