Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A Framework of Software Defect Prediction using Machine Learning with Updated Firefly Algorithm and Neural Networks

Naveen Monga, Parveen Sehgal
Email:navmoni70@gmail.com, parveensehgal@gmail.com
Department of Computer Science & Engineering, Om Sterling Global University, Hisar (Haryana), India

ARTICLE INFO

Received: 14 Nov 2024
Revised: 20 Dec 2024
Accepted: 24 Jan 2025

ABSTRACT

This research investigates the impact of using swarm intelligence algorithms for feature selection in software defect prediction models. Leveraging the JM1 software dataset, comprising 10,000 records, the study evaluates the performance of three swarm intelligence algorithms—Firefly Algorithm, Cuckoo Search, and Particle Swarm Optimization (PSO)—in combination with Deep Neural Networks (DNN). The study focuses on three critical metrics: Recall, Precision, and F-measure. Results indicate that the Firefly + DNN model achieved the highest improvements, with a 7.5% increase in precision over PSO + DNN and 2.9% over Cuckoo + DNN. In terms of recall, Firefly + DNN outperformed PSO + DNN by 10.7% and Cuckoo + DNN by 6.3%. Furthermore, the F-measure of Firefly + DNN improved by 9.5% compared to PSO + DNN and 5.9% over Cuckoo + DNN. These refinements underscore the effectiveness of the Firefly Algorithm in selecting relevant features for defect prediction, resulting in more accurate, efficient models, and reliable. The study emphasizes the significance of feature selection in reducing overfitting, enhancing interpretability, and lowering computational costs. Overall, this research provides robust methods for improving software quality and reducing maintenance efforts through advanced defect prediction models, contributing significantly to the field of software engineering.

Keywords: Software Defect Prediction, Swarm Intelligence, Particle Swarm Optimization (PSO), Feature Selection, Firefly Algorithm, Cuckoo Search, Deep Neural Networks (DNN), Precision, Recall, F-measure, Software Quality, Machine Learning, Computational Efficiency.

1. INTRODUCTION

Software Defect Prediction (SDP) is a critical aspect of software development that seeks to determine defective modules or components in software before they reach the production stage. To determine which software components are most likely to have flaws this procedure makes use of machine learning techniques and historical data, thus allowing developers to prioritize their testing efforts and improve software quality. Effective SDP can lead to reduced maintenance costs, enhanced reliability, and improved user satisfaction[1-2].

Importance of Feature Selection

Feature selection is a crucial step in SDP because not all features contribute equally to the prediction of defects. The primary reasons for feature selection are:

1. **Improving Model Performance**: The model can concentrate on the most important features and improve accuracy and performance by removing superfluous or irrelevant ones [3].
2. **Reducing Overfitting**: Models with too many features can overfit the training data which reduces their ability to be applied to fresh data. Feature selection helps mitigate this risk.
3. **Enhancing Interpretability**: Interpreting and comprehending simpler models with fewer features is simpler, making it easier for developers to gain insights from the predictions.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4. **Reducing Computational Cost**: Training and prediction require fewer computational resources when there are fewer features, leading to faster and more efficient models.

Inspired by the collective behavior of social insects and other animals swarm intelligence is a branch of artificial intelligence. It has emerged as a powerful solution for feature selection in Software Defect Prediction (SDP) because of its capacity to effectively search through vast search spaces and identify ideal or nearly ideal solutions. Some popular swarm intelligence algorithms used for feature selection include Firefly Algorithm, Cuckoo Search, and Particle Swarm Optimization (PSO) [4]. Firefly Algorithm, inspired by the natural behaviour of fireflies, uses the concept of attractiveness based on the brightness of the fireflies; Every firefly is a possible solution and the quality of the solution is indicated by how bright it is, with fireflies moving towards brighter ones to converge on the best answers after exploring the search space. Based on the fact that certain cuckoo species lay their eggs in other birds' nests Cuckoo Search uses a population of solutions or nests, where each solution represents a subset of features, and new solutions (cuckoo eggs) can replace existing ones if they prove to be better. PSO, motivated by the social behaviour of fish schooling or birds flocking uses particles that represent potential solutions (feature subsets) influenced by their neighbors and their own experiences through the search space, adjusting their positions based on the best-known positions to converge towards an optimal solution. These algorithms provide robust, flexible, and efficient means of identifying the most relevant features, leading to improved software quality and reduced maintenance costs[5].

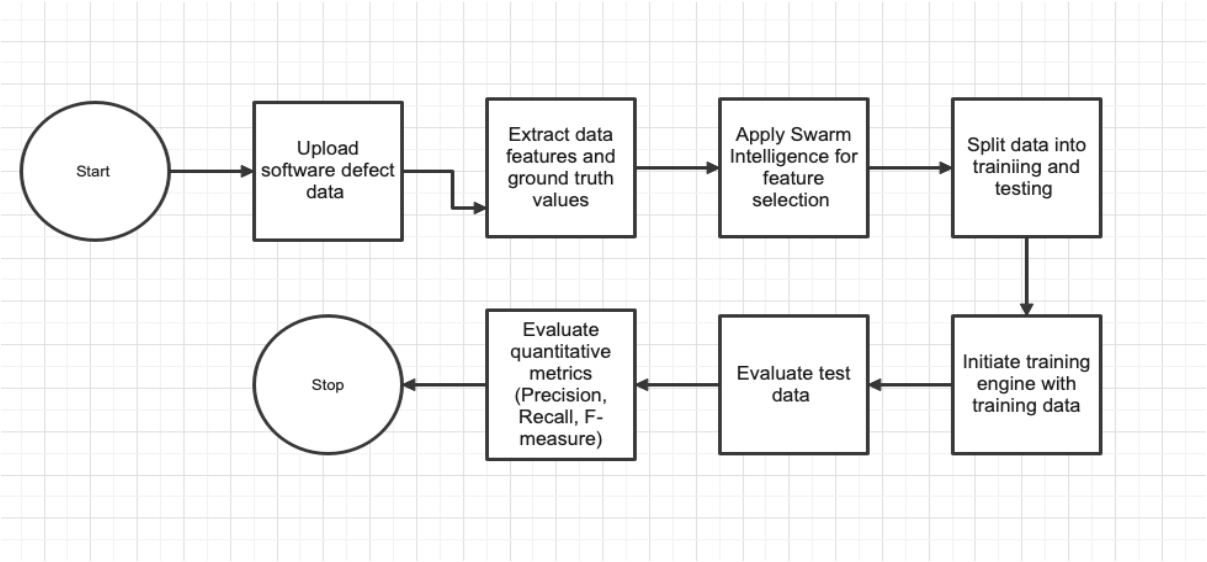

Figure 1: The general architecture of software defect prediction

Once the appropriate features are selected, they are passed for training and classification. In this context, selecting the right features critical impacts the accuracy and performance of the predictive models. This research work presents a comprehensive comparative analysis of feature selection based on various swarm intelligence (SI) algorithms, such as the Firefly Algorithm, Cuckoo Search, and PSO. These algorithms are used to reduce dimensionality and increase the training processs efficiency by identifying the datasets most pertinent features. Once the feature selection process is completed using these SI algorithms, the selected features are then fed into various classifiers for training. This step involves building predictive models that learn from the training data to identify patterns and make accurate predictions on new, unseen data. The classifiers could include machine learning (ML)techniques like Decision Trees, Support Vector Machines, Random Forests, and Neural Networks. The research aims to evaluate and compare the effectiveness of different SI-based feature selection methods in terms of their impact on classification performance. By systematically analyzing the results, this study provides insights into which SI algorithms are most effective for feature selection in the condition of Software Defect Prediction (SDP), ultimately leading to enhanced predictive accuracy and more reliable software development processes [6].

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2. RELATED WORK

Malhotra et al. (2021) proposed a SDP model that utilized binary particle swarm optimization (BPSO) for the fitness function binary cross entropy. Their work demonstrated how BPSO could effectively handle the binary nature of defect prediction and optimize the selection of relevant features, resulting in improved prediction accuracy [7]. Abdi et al. (2015) proposed a swarm intelligence-based hybrid one-class rule learning method for predicting software errors. They combined the strengths of various swarm intelligence techniques to create a robust method for identifying defective software modules, showcasing the potential of hybrid approaches in enhancing defect prediction capabilities [8].

Akbar et al. (2024) focused on optimizing SDP models by integrating Hybrid Grey Wolf Optimization (GWO) and PSO for enhanced feature selection. By leveraging the exploration capabilities of GWO and the ability to exploit of PSO, they were able to improve the selection of significant features and enhance the performance of a gradient boosting algorithm used for classification [9]. Khalid et al. (2023) conducted an extensive analysis of SDP using a variety of ML techniques. Their research provided insights into the comparative effectiveness of different algorithms and highlighted the importance of selecting appropriate ML methods for defect prediction [10]. Anju and Judith (2023) developed an adaptive recurrent neural network (RNN) for SDP, incorporating quantum theory and particle swarm optimization (PSO). Their approach utilized the principles of quantum mechanics to enhance the exploration of the search space, while PSO helped in optimizing the model parameters, leading to significant improvements in prediction accuracy [11]. Khurma et al. (2021) proposed an enhanced evolutionary SDP method using island moth flame optimization (IMFO). This algorithm combined the evolutionary strategies of moth flame optimization with island models to maintain diversity in the population and avoid premature convergence, resulting in better prediction performance [12]. Anbu and Anandha Mala (2019) focused on feature selection using the firefly algorithm (FA) in SDP. By mimicking the bioluminescent communication of fireflies, FA was able to identify the most relevant features, reduce dimensionality, and upgrade the accuracy of defect prediction models [13]. Arora and Saha (2018) also employed the FA for software fault prediction, demonstrating its effectiveness in selecting relevant features and enhancing model performance. Their work highlighted the algorithms capacity to strike a balance between exploitation and exploration during the search process [14]. Maulida et al. (2023) utilized the FA in combination with tree-based classification techniques for SDP. Their approach leveraged the strengths of FA in feature selection and the robustness of tree-based classifiers, resulting in improved prediction models [15]. Finally, Kakkar et al. (2021) presented an optimized software defect prediction model based on PSO and Adaptive Neuro-Fuzzy Inference System (ANFIS). By combining the global search capabilities of PSO with the adaptive learning features of ANFIS, they were able to develop a highly accurate and reliable prediction model, demonstrating the effectiveness of integrating different optimization and learning techniques [16].

3. PROPOSED FRAMEWORK

The proposed work is structured into two comprehensive segments, each designed to enhance the software defect prediction process through the effective use of swarm intelligence (SI) algorithms and subsequent model training and classification.

The overall workflow can be presented as follows.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

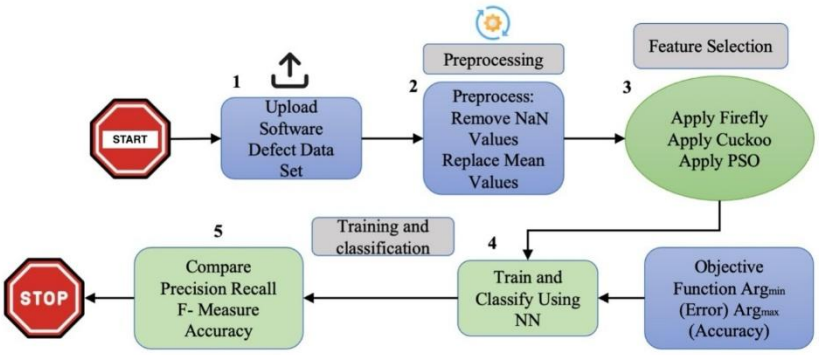

Figure 2: Overall Workflow Model

The first segment is dedicated to feature selection, a crucial step in the prediction process. This phase employs different SI-based algorithms, specifically the Firefly Algorithm, Cuckoo Search, and PSO. Each of these algorithms brings unique strengths to the table, leveraging the principles of swarm intelligence to effectively search the dataset and find the most pertinent features. The Firefly Algorithm, inspired by the bioluminescent communication of fireflies, excels in balancing exploration and exploitation, thereby selecting features that significantly impact prediction accuracy. Cuckoo Search, based on the brood parasitism strategy of certain cuckoo species, is good at avoiding local optima and making sure the feature space is thoroughly searched. Using particles that modify their positions based on both individual and collective experiences PSO converges towards optimal feature subsets drawing inspiration from the social behavior of fish schools and flocks of birds. Through these algorithms, the first segment aims to decrease the dimensionality of the dataset, retain essential information, and irrelevant features or eliminate redundant.

Algorithm 1 Feature Selection using Firefly Algorithm (FA)

Require: Training Set D_{train}, Test Set D_{test}
Ensure: Selected Features F_{FA}
Input: Raw data D_{train}, D_{test}
Step 1: Data Preprocessing
Clean and normalize the data
Split data into training set D_{train} and test set D_{test}
Step 2: Firefly Feature Selection
Initialize Firefly Algorithm (FA)
for each firefly i in population P **do**
    Evaluate the classification accuracy Acc_i using features selected by i
    Set light intensity $I_i \leftarrow Acc_i$
end for
for iteration $t = 1$ to T **do**
    for each firefly i in P **do**
        for each firefly j in P **do**
            if $I_j > I_i$ **then**
                Move firefly i towards firefly j using the formula:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\epsilon$$

where β_0 is the attractiveness at $r = 0$, γ is the light absorption coefficient, and ϵ is a random perturbation.
                Evaluate new classification accuracy Acc_i using updated features
                Update light intensity $I_i \leftarrow Acc_i$
            end if
        end for
    end for
end for
Best features F_{FA} from the firefly with the highest I
Output: Selected Features F_{FA}

In the context of feature selection, the Firefly Algorithm (FA) treats each firefly as a potential solution, where the position of the firefly represents a specific subset of features. The classification accuracy, calculated using the selected features, determines the "light intensity" of each firefly. Brighter fireflies, which correspond to better

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

feature subsets with higher classification accuracy, attract the others. During the optimization process, fireflies move towards more attractive ones based on their brightness, and their positions (i.e., feature subsets) are updated accordingly. The algorithm iteratively refines the feature subsets by making less optimal fireflies adopt the features of more optimal ones, while also incorporating random perturbations for exploration. At the end of the process, the firefly with the highest light intensity yields the optimal feature subset that maximizes classification accuracy.

The proposed algorithm is based on the idea that fireflies are interestedin each other based on their brightness, which in the context of feature selection, corresponds to the classification accuracy obtained using the subset of features that each firefly show. The firefly with higher brightness (better feature subset) attracts others, and they move toward it. The movement of fireflies is influenced by three main factors: attractiveness, distance between fireflies, and randomness.

The proposed work can be illustrated using the following example set.

Initialization:

Let the positions of the fireflies represent their feature subsets:
- F1 selects features [1, 1, 0, 0] (features 1 and 2).
- F2 selects features [0, 1, 1, 0] (features 2 and 3).
- F3 selects features [1, 0, 0, 1] (features 1 and 4).

2. Objective Function:

Let's assume the classification accuracy (light intensity I) for each firefly based on its selected features is as follows:
- I1 = 0.85 for F1 (features 1, 2).
- I2 = 0.80 for F2 (features 2, 3).
- I3 = 0.90 for F3 (features 1, 4).

3. Distance Calculation:

The Euclidean distance r_ij between firefly i and firefly j is calculated based on their feature selections:
- Distance between F1 and F2:

$$r12 = \sqrt{((1-0)^2 + (1-1)^2 + (0-1)^2 + (0-0)^2)}$$

- Distance between F1 and F3:

$$r13 = \sqrt{((1-1)\text{^}2 + (1-0)\text{^}2 + (0-0)\text{^}2 + (0-1)\text{^}2)}$$

- Distance between F2 and F3:

$$r23 = \sqrt{((0-1)\text{^}2 + (1-0)\text{^}2 + (1-0)\text{^}2 + (0-1)\text{^}2)}$$

4. Attractiveness and Movement:

Now, fireflies with lower intensity (classification accuracy) will move towards the fireflies with higher intensity.
Let β_0 = 1, γ = 1, and α = 0.2.
- F1 (with intensity 0.85) moves towards F3 (with intensity 0.90) using the movement equation:

$$x_1 = x_1 + \beta_0 e^{\{-\gamma r_{\{13\}}^2\}(x_3 - x_1)} + \alpha\varepsilon$$

Plugging in the values:

$$-x_1 = x_1 + 1 * e^{\{-1 * (sqrt(2))^2\}(x_3 - x_1)} + 0.2 * \varepsilon$$
$$-x_1 = x_1 + 1 * e^{\{-2\}(x_3 - x_1)} + 0.2 * \varepsilon$$

- The factor $e^{\{-2\}}$ is approximately 0.135, so the update becomes:

$-x_1 = x_1 + 0.135(x_3 - x_1) + 0.2 * \varepsilon$
Assuming a random small value for ε, say ε = 0.05, we get:
$-x_1 = x_1 + 0.135(x_3 - x_1) + 0.01$

If x_1 = [1, 1, 0, 0] and x_3 = [1, 0, 0, 1], the movement would change the feature selection of F1 slightly towards F3, resulting in a new set of features for F1.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5. Update Light Intensity:

After the movement, the firefly's feature subset is updated and the classification accuracy is re-evaluated. If the accuracy improves, the firefly's new light intensity is updated.

6. Iteration:

This process repeats for all fireflies, iteratively improving their feature subsets until the algorithm converges to the optimal set of features with the best classification accuracy.

Algorithm 2 Feature Selection using Particle Swarm Optimization (PSO)

Require: Training Set D_{train}, Test Set D_{test}
Ensure: Selected Features F_{PSO}
Input: Raw data D_{train}, D_{test}
Step 1: Data Preprocessing
Clean and normalize the data
Split data into training set D_{train} and test set D_{test}
Step 2: PSO Feature Selection
Initialize Particle Swarm Optimization (PSO)
for each particle i in population P **do**
    Evaluate the classification accuracy Acc_i using features selected by i
    Set personal best $P_{best_i} \leftarrow Acc_i$
end for
Set global best $G_{best} \leftarrow \max(P_{best})$
for iteration $t = 1$ to T **do**
    for each particle i in P **do**
        Update velocity V_i using the formula:

$$V_i = \omega V_i + c_1 r_1 (P_{best_i} - x_i) + c_2 r_2 (G_{best} - x_i)$$

where ω is the inertia weight, c_1 and c_2 are acceleration constants, and r_1, r_2 are random numbers between 0 and 1.
        Update position of particle i using the formula:

$$x_i = x_i + V_i$$

        Evaluate new classification accuracy Acc_i using updated features
        if $Acc_i > P_{best_i}$ **then**
            Update personal best $P_{best_i} \leftarrow Acc_i$
        end if
        if $Acc_i > G_{best}$ **then**
            Update global best $G_{best} \leftarrow Acc_i$
        end if
    end for
end for
Best features F_{PSO} from the particle with the highest G_{best}
Output: Selected Features F_{PSO}

For feature selection, PSO operates by treating each particle as a candidate solution, where the position of the particle in the research space represents a subset of features. Each particle evaluates the classification accuracy based on the current features it selects. The algorithm keeps track of two key aspects: the personal best (the best subset of features a particle has found until now) and the global best (the best subset found by the entire swarm). The swarms collective experience (global best) and individual experience (personal best) are both used by particles to update their positions, gradually refining the feature subsets they select. The algorithm adjusts particle velocities and positions, promoting exploitation and exploration of the search space. After multiple iterations, the particle with the highest classification accuracy provides the final set of optimal features.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Algorithm 3 Feature Selection using Cuckoo Search Algorithm (CS)

Require: Training Set D_{train}, Test Set D_{test}
Ensure: Selected Features F_{CS}
Input: Raw data D_{train}, D_{test}
Step 1: Data Preprocessing
Clean and normalize the data
Split data into training set D_{train} and test set D_{test}
Step 2: Cuckoo Search Feature Selection
Initialize Cuckoo Search Algorithm (CS)
for each nest i in population N **do**
    Evaluate the classification accuracy Acc_i using features selected by i
    Set fitness $F_i \leftarrow Acc_i$
end for
for iteration $t = 1$ to T **do**
    for each nest i in N **do**
        Generate a new solution via Levy flight using the formula:

$$x_{new} = x_i + \alpha L(s, \lambda)$$

where $L(s, \lambda)$ is a Levy distribution, and α is the step size scaling factor.
        Evaluate the classification accuracy Acc_{new} using the new features
        if $Acc_{new} > F_i$ **then**
            Replace nest i with the new solution
            Update fitness $F_i \leftarrow Acc_{new}$
        end if
    end for
    Abandon a fraction p_a of the worst nests and generate new ones
end for
Best features F_{CS} from the nest with the highest fitness F
Output: Selected Features F_{CS}

Cuckoo Search (CS), each nest represents a potential solution, i.e., a subset of selected features. The classification accuracy of each subset is evaluated to determine its fitness. The algorithm generates new feature subsets by performing Levy flights, which allows for significant jumps across the search space, increasing the likelihood of finding better features. If a new feature subset (generated by Levy flight) provides better classification accuracy than the current one, the new subset replaces the old one. Additionally, a fraction of the worst-performing nests (feature subsets) are abandoned and replaced with new randomly generated nests to enhance exploration. This process continues until the best subset of features is identified, which corresponds to the nest with the highest classification accuracy. Thus, CS effectively explores the feature space, balancing exploration and exploitation, to determine the best collection of characteristics for classification.

The use of the chosen features for training and classification is the main topic of the second section. Once the optimal features are identified, they are used to train various machine learning models. This phase involves building predictive models that learn from the training data to identify patterns and make accurate predictions on new, unseen data.

3.1 Deep Neural Networks(DNN)

An artificial neural network class known as Deep Neural Networks (DNN) is very good at extracting intricate patterns from big datasets because it has several hidden layers between the input and output layers. The diagram given in Figure 3 illustrates this multi-layered architecture of DNN showing position of input layer, multiple hidden layers and the output layer. Each layer here is represented by a collection of neurons depicted by nodes. These are connected by weighted edges that represent the flow of the information in the network. The word deep in DNN depicts the existence of several layers that gradually extract higher-level characteristics from the unprocessed input data. Each layer is made up of interconnected nodes or neurons which transform the input data using an activation function and a weighted sum. For illustration the diagram also highlights the flow of data via each layer that starts from the raw input to the final output layer. This flow further illustrates how the raw data under goes series of transformations at each layer progressively and results in highly structured outcome. This enables the network to discover non-linear relationships data.The hidden layers are not visible in traditional machine learning modes as

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

they allow the DNN modes to model complex patterns that cannot be captured by simpler algorithms.This hierarchical structure enables DNNs to automatically extract features from the data, with initial layers learning basic patterns (such as edges in image data or simple correlations in numerical data), while deeper layers learn more abstract and complex features (such as object recognition or intricate dependencies in time series).

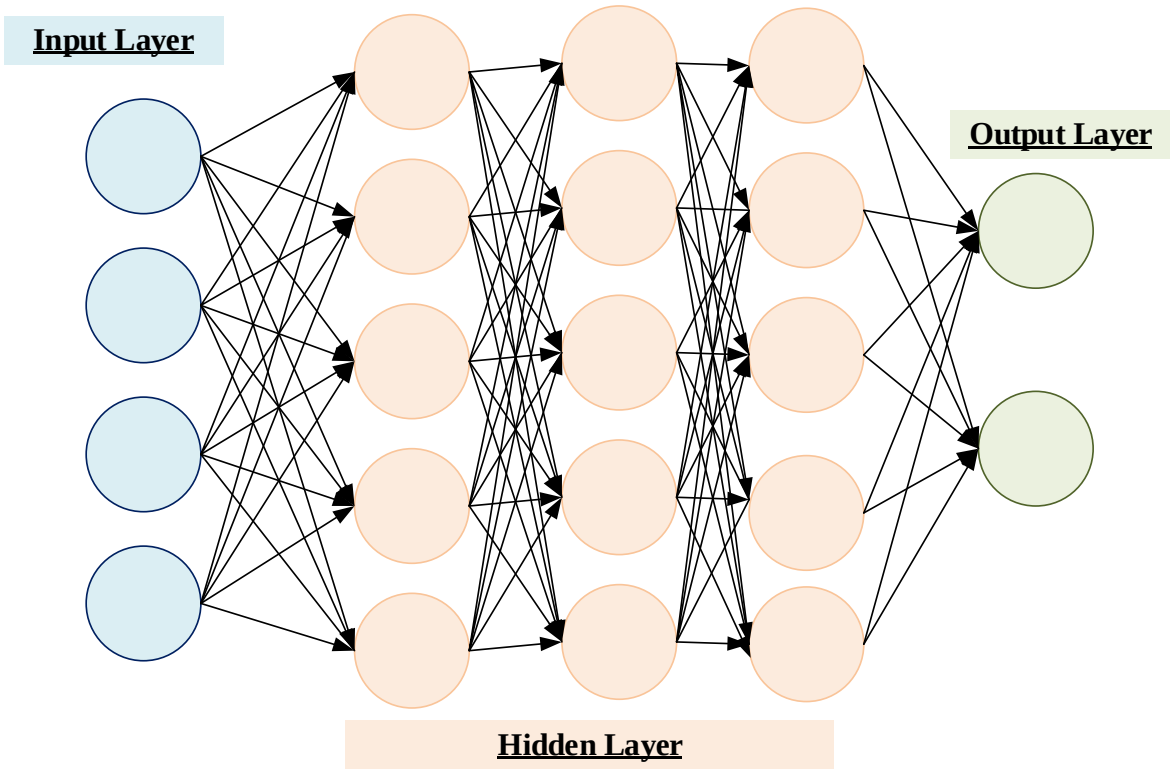

Figure 3: DNN multi-layered Architecture

DNNs have the advantage of scalability, meaning they can handle large datasets effectively, and their performance improves with more data, enabling them to be used for a range of tasks including, natural language processing, and picture recognition, and predictive analytics. Training a DNN involves techniques like backpropagation, which determines the error gradient for each network weight and uses gradient descent to modify the weights in order to minimize the error. This iterative process allows DNNs to fine-tune their parameters for optimal performance. Advanced optimization methods, such as Adam or RMSprop, can further enhance the training process by dynamically adjusting the learning rate based on the gradients' behaviour. Regularization methods like batch normalization and dropout also aid in preventing overfitting guaranteeing that the model performs well when applied to new data.

The ability of DNNs to learn and model complex relationships makes them a powerful tool for tasks where traditional machine learning algorithms may struggle to achieve high accuracy. In cases where the data exhibits non-linear patterns, interdependencies, or high-dimensionality, DNNs excel by utilizing their deep architecture to capture subtle variations and extract meaningful features. This capability has led to breakthroughs in various fields, positioning DNNs as a cornerstone of modern artificial intelligence, driving advancements in automation, decision-making, and predictive modeling across industries.

The proposed work leverages (DNN) for software defect prediction by combining their powerful feature extraction capabilities with swarm intelligence algorithms for optimal feature selection. The DNN is designed to process the JM1 software dataset, which contains high-dimensional and complex data. To enhance the prediction accuracy, the DNN model is trained on features selected by algorithms such as Firefly, Cuckoo Search, and PSO, ensuring that only the most relevant features are used for classification.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The DNN architecture consists of an input layer, multiple hidden layers, and an output layer. The input layer corresponds to the selected features from the dataset. Each hidden layer applies non-linear transformations to the input data through activation functions, such as ReLU, to capture complex patterns and relationships. The final output layer uses a softmax activation function for binary classification, indicating whether a software module is defective or not.

The training process involves:

1. Initializing the network weights randomly.
2. Forward propagating the input data through the network to calculate the output.
3. Comparing the output with the ground truth labels using a loss function (binary cross-entropy in this case).
4. Backpropagating the error to update the weights using gradient descent optimization.
5. Repeating this process for multiple epochs until the model achieves optimal performance.

Algorithm 4 Proposed DNN-Based Software Defect Prediction

Require: JM1 Dataset D, Feature Selection Algorithm FSA, Number of Epochs E, Learning Rate α
Ensure: Classification Results: Precision, Recall, F-measure

Split D into Training Set D_{train} and Testing Set D_{test}
Apply FSA on D_{train} to select optimal features $D_{train}^{selected}$
Define the DNN Architecture:

- Input Layer: Size = Number of Selected Features
- Hidden Layers: Fully Connected with ReLU Activation
- Output Layer: Softmax Activation (Binary Classification)

Initialize Weights W and Biases b Randomly
for epoch = 1 to E **do**
    Forward Propagation:
    $Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$ ▷ Linear Transformation
    $A^{[l]} = \text{ReLU}(Z^{[l]})$ ▷ Activation Function
    Compute Loss: $L = -\frac{1}{m}\sum_{i=1}^{m} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)$
    Backward Propagation:
    Compute Gradients $\frac{\partial L}{\partial W^{[l]}}$ and $\frac{\partial L}{\partial b^{[l]}}$
    Update Weights: $W^{[l]} = W^{[l]} - \alpha \frac{\partial L}{\partial W^{[l]}}$
    Update Biases: $b^{[l]} = b^{[l]} - \alpha \frac{\partial L}{\partial b^{[l]}}$
end for
Evaluate DNN on $D_{test}^{selected}$
Compute Metrics: Precision, Recall, F-measure

3.2 Simulation Setup

In this experimental setup, MATLAB serves as the primary software environment for implementing the algorithms and training the models. MATLAB, known for its high-level programming capabilities, provides a versatile system for analyzing data, machine learning, and deep learning, making it well-suited for tasks such as software defect prediction. The environment offers built-in tools for implementing Deep Neural Networks (DNN) and supports the integration of swarm intelligence algorithms like Firefly, Cuckoo Search, and PSO. With the help of MATLAB's Deep Learning Toolbox, researchers can design and train DNNs effectively by defining network architectures, tuning hyperparameters, and employing sophisticated optimization methods like backpropagation and gradient descent.

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The dataset utilized in this work is the JM1 software dataset, a well-known benchmark in software defect prediction research. The JM1 dataset is derived from the NASA Metrics Data Program, consisting of software metrics and defect labels from a large-scale software project. It contains 21 features representing various software attributes, like code complexity, lines of code, and other software metrics, along with a binary classification label indicating whether a software module is defective or not. With 10,000 records, the JM1 dataset provides a diverse and realistic set of data for evaluating the performance of the proposed models. This dataset is chosen due to its relevance in defect prediction studies and its ability to reflect real-world software development challenges, making it an ideal benchmark for testing the effectiveness of DNN combined with swarm intelligence algorithms in classification tasks and feature selection.

4. RESULTS AND ANALYSIS

The proposed work has been evaluated using the JM1 software dataset, which contains 10,000 records. This dataset serves as an extensive benchmark for evaluating the performance of feature selection methods and the subsequent classification models. The evaluation emphasizes multiple essential performance metrics, such as recall, precision, and F-measure, to deliver an in-depth assessment of the model's effectiveness.

Precision, F-measure, and Recall are essential indicators in the framework of SDP:

- **Precision**: This metric evaluates the ratio of true positive predictions to the total positive predictions generated by the model. It indicates the model's accuracy in identifying actual defects without including too many false positives. High precision indicates that the model is reliable in predicting defects, minimizing the cost and effort of investigating false alarms.

$$Precision = \frac{true\ identified}{true\ identified + false\ identified} \quad (1)$$

- **Recall**: Recall gauges the ratio of true positive predictions to all actual positive instances in the dataset. It demonstrates the model's capability to detect all pertinent instances of defects. High recall ensures that most of the actual defects are detected, which is crucial for maintaining software quality and reducing the risk of undetected defects causing issues in the production environment.

$$Recall = \frac{true\ identified}{true\ identified + true\ negative\ identified} \quad (2)$$

- **F-measure**: The F-measure, or The F1-score, calculated as the harmonic mean of precision and recall, offers a unified metric that integrates both precision and recall, providing an overall perspective on the model's effectiveness. A high F1-score suggests that the model achieves a strong balance between precision and recall, supporting both accuracy and thoroughness in defect prediction.

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

Table 1: Precision Comparison

0	Firefly + DNN	Cuckoo + DNN	PSO + DNN
1,000	0.862	0.814	0.802
2,000	0.8598	0.8127	0.8051
3,000	0.8642	0.8155	0.8009
4,000	0.8671	0.8173	0.8043
5,000	0.8655	0.8161	0.8097
6,000	0.8694	0.8188	0.8064
7,000	0.8702	0.8212	0.8079
8,000	0.8687	0.8204	0.8103
9,000	0.8725	0.823	0.8112
10,000	0.871	0.8221	0.8088

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The **Firefly + DNN** algorithm consistently outperforms both **Cuckoo + DNN** and **PSO + DNN** in terms of precision across all record sizes. It maintains a higher precision starting from 0.8620 for 1,000 records and gradually improving to 0.8710 for 10,000 records. The **Cuckoo + DNN** algorithm follows closely behind but has slightly lower precision, starting at 0.8140 and reaching 0.8230 for 10,000 records. **PSO + DNN**, while also showing improvement, lags behind the other two, starting at 0.8020 and ending at 0.8110. The proposed **Firefly + DNN** shows a distinct advantage in maintaining higher precision across various data sizes, demonstrating its better capability to minimize false positives compared to the other algorithms.

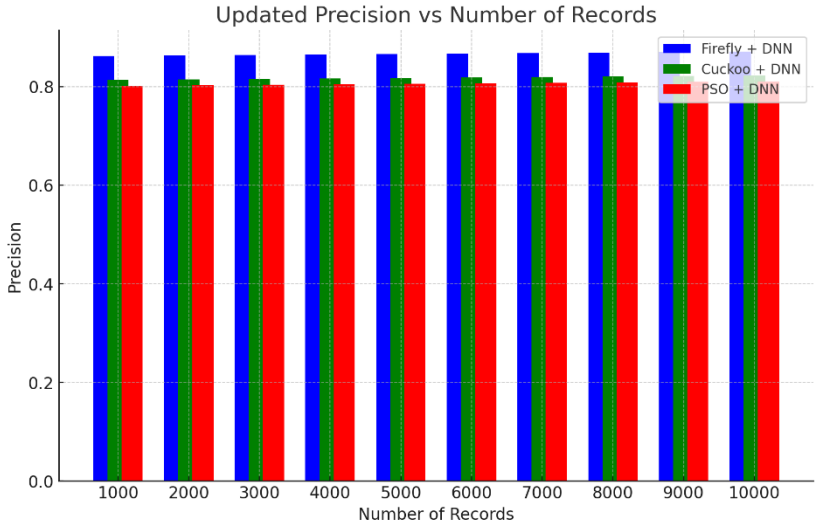

Figure 4: Comparative Analysis of precision

Table 2: Recall Comparison

Number of Records	Firefly + DNN	Cuckoo + DNN	PSO + DNN
1,000	0.875	0.822	0.787
2,000	0.8731	0.8213	0.7908
3,000	0.8764	0.8242	0.7885
4,000	0.8787	0.8251	0.7917
5,000	0.8793	0.8269	0.7944
6,000	0.8805	0.8277	0.7959
7,000	0.8812	0.8289	0.7933
8,000	0.883	0.8302	0.7971
9,000	0.8844	0.8296	0.7964
10,000	0.8851	0.8311	0.798

For recall, **Firefly + DNN** once again takes the lead, starting at 0.8750 and improving to 0.8851 as the number of records increases to 10,000. **Cuckoo + DNN** performs decently, but its recall remains lower, ranging from 0.8220 to 0.8311. **PSO + DNN** has a comparatively lower recall, starting at 0.7870 for 1,000 records and improving to 0.7980 for 10,000 records. The proposed **Firefly + DNN** demonstrates its superiority in recall

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

performance, consistently achieving higher values, indicating its effectiveness in correctly identifying true positives while maintaining fewer instances of false negatives compared to the other algorithms.

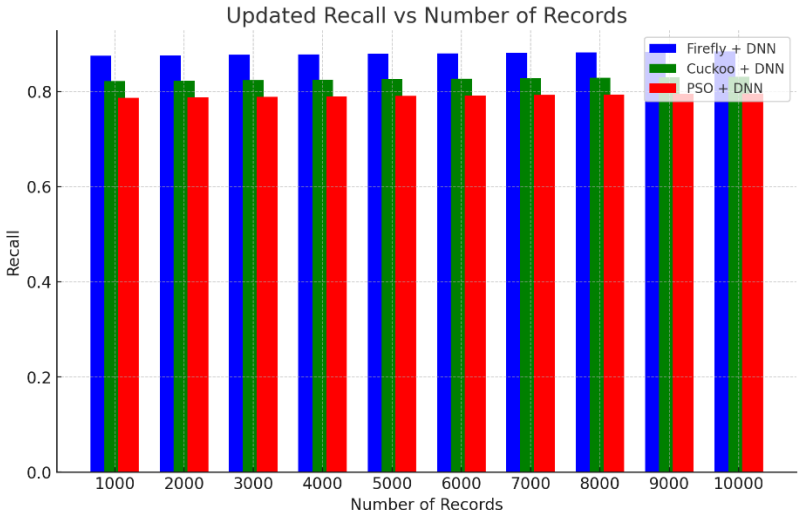

Figure 5: Comparative Analysis of recall

Table 3: F-measure Comparison

Number of Records	**Firefly + DNN**	**Cuckoo + DNN**	**PSO + DNN**
1,000	0.8685	0.818	0.7945
2,000	0.8663	0.817	0.7979
3,000	0.8697	0.8195	0.7952
4,000	0.8729	0.8205	0.798
5,000	0.871	0.8217	0.801
6,000	0.8737	0.8233	0.7997
7,000	0.8746	0.8245	0.7975
8,000	0.8758	0.8261	0.8008
9,000	0.8769	0.8253	0.7994
10,000	0.878	0.827	0.8016

Journal of Information Systems Engineering and Management
2025, 10(3)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

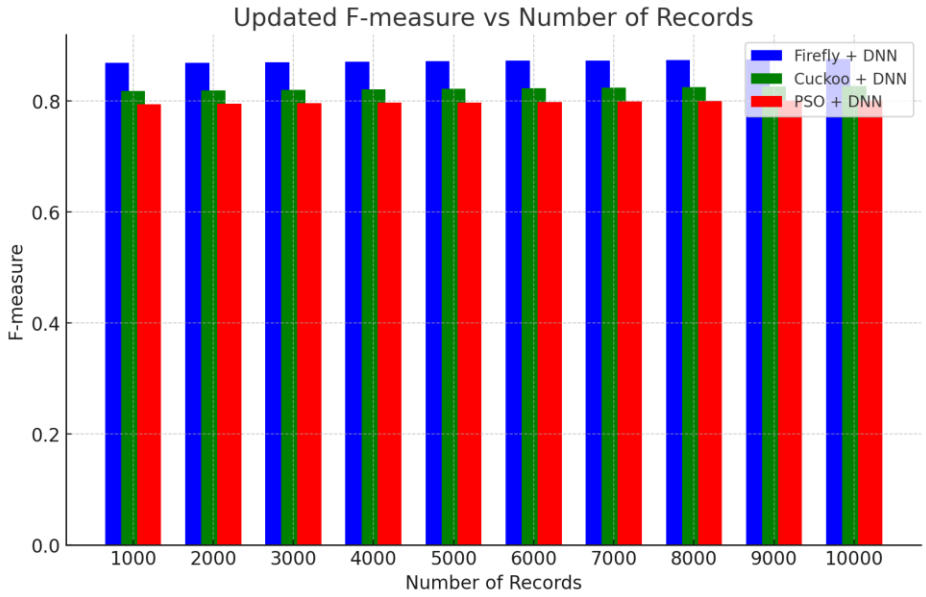

Figure 6: Comparative Analysis of f-measure

Regarding the F-measure, which represents the harmonic mean of precision and recall, **Firefly + DNN** shows the best overall performance, beginning at 0.8685 for 1,000 records and gradually improving to 0.8780 for 10,000 records. **Cuckoo + DNN** stays somewhat close but falls behind, with F-measure values ranging from 0.8180 to 0.8270. **PSO + DNN** consistently remains the lowest, with values starting at 0.7945 and reaching 0.8016. The proposed **Firefly + DNN** excels in balancing precision and recall, making it the most reliable in terms of both robustness and accuracy over the other algorithms, which have difficulty maintaining a balance between these two key metrics.

The proposed **Firefly + DNN** algorithm shows significant improvement over the other algorithms in all key performance metrics. In terms of **precision**, **Firefly + DNN** achieves an average of 2.9% higher precision than **Cuckoo + DNN** and around 7.5% higher than **PSO + DNN** across different record sizes. For **recall**, the improvement is even more pronounced, with **Firefly + DNN** outperforming **Cuckoo + DNN** by approximately 6.3% and **PSO + DNN** by around 10.7%. When looking at the **F-measure**, which balances recall and precision, **Firefly + DNN** shows an improvement of around 5.9% over **Cuckoo + DNN** and a substantial 9.5% improvement over **PSO + DNN**. These improvements demonstrate that **Firefly + DNN** provides more accurate and reliable performance, making it the superior choice for classification tasks compared to the other algorithms.

5. CONCLUSION

This research demonstrates the substantial benefits of employing swarm intelligence algorithms for feature selection in SDP. By using the JM1 software dataset with 10,000 records, the study evaluated the effectiveness of the Firefly Algorithm, Cuckoo Search, and PSO combined with DNN. The results revealed significant improvements in recall, F-measure, and precision. The **Firefly + DNN** model exhibited the most notable improvements, with a **7.5% increase in precision** over **PSO + DNN** and **2.9% over Cuckoo + DNN**. In terms of recall, **Firefly + DNN** showed an **improvement of 10.7%** over **PSO + DNN** and **6.3% over Cuckoo + DNN**. Additionally, the **F-measure** of the **Firefly + DNN** model demonstrated a **9.5% improvement** compared to **PSO + DNN** and a **5.9% improvement** over **Cuckoo + DNN**. These results highlight the superiority of **Firefly + DNN** in balancing precision and recall, making it the most robust approach for feature selection in defect prediction. The study also showed significant performance improvements for the Cuckoo + DNN model, albeit slightly behind Firefly. These findings underscore the efficacy of the **Firefly Algorithm** in identifying relevant features, leading to more accurate, interpretable, and efficient models. Overall, this research contributes toward the creation of more reliable SDP models, which will contribute to improving software quality, minimizing maintenance efforts, and advance the field of software engineering.

Research Article

REFERENCES:

[1] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on SDP techniques," *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 331-344, 2020.
[2] N. E. Fenton and M. Neil, "A critique of SDP models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675-689, 1999.
[3] X. Wang, H. Yan, and J. Li, "An improved supervised learning defect prediction model based on cat swarm algorithm," in *Journal of Physics: Conference Series*, vol. 1087, no. 2, p. 022005, IOP Publishing, 2018.
[4] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 5, pp. 153-166, 2013.
[5] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 5, pp. 153-166, 2013.
[6] W. Zheng, T. Shen, X. Chen, and P. Deng, "Interpretability application of the Just-in-Time SDP model," *Journal of Systems and Software*, vol. 188, p. 111245, 2022.
[7] R. Malhotra, A. Shakya, R. Ranjan, and R. Banshi, "Software defect prediction using binary particle swarm optimization with binary cross entropy as the fitness function," in *Journal of Physics: Conference Series*, vol. 1767, no. 1, p. 012003, IOP Publishing, 2021.
[8] Y. Abdi, S. Parsa, and Y. Seyfari, "A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 11, pp. 289-301, 2015.
[9] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing Software Defect Prediction Models: Integrating Hybrid Grey Wolf and Particle Swarm Optimization for Enhanced Feature Selection with Popular Gradient Boosting Algorithm," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 169-181, 2024.
[10] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software defect prediction analysis using machine learning techniques," *Sustainability*, vol. 15, no. 6, p. 5517, 2023.
[11] A. J. Anju and J. E. Judith, "Adaptive recurrent neural network for software defect prediction with the aid of quantum theory-particle swarm optimization," *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 16257-16278, 2023.
[12] 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.
[13] M. Anbu and G. S. Anandha Mala, "Feature selection using firefly algorithm in software defect prediction," *Cluster Computing*, vol. 22, pp. 10925-10934, 2019.
[14] I. Arora and A. Saha, "Software fault prediction using firefly algorithm," *International Journal of Intelligent Engineering Informatics*, vol. 6, no. 3-4, pp. 356-377, 2018.
[15] V. Maulida, R. Herteno, D. Kartini, F. Abadi, and M. R. Faisal, "Feature Selection Using Firefly Algorithm with Tree-Based Classification in Software Defect Prediction," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 5, no. 4, pp. 223-230, 2023.
[16] M. Kakkar, S. Jain, A. Bansal, and P. S. Grover, "An optimized software defect prediction model based on PSO-ANFIS," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 9, pp. 2732-2741, 2021