A Practical Method for Software Defect Prediction Using Swarm-Based Optimization Techniques

A S Karthikeyan[1], K Yatheendra[2], K Bhaskar[3]

[1] P.G Scholar, Department of MCA, Sri Venkatesa Perumal College of Engineering & Technology, Puttur, E-mail: askarthikeyan2002@gmail.com, ORC-ID: https://orcid.org/0009-0007-0941-4188

[2]Assistant Professor, Department of CSE(AI & ML), Sri Venkatesa Perumal College of Engineering & Technology, Puttur, E-mail: k.yatheendra84@gmail.com, ORC-ID: https://orcid.org/0009-0003-1382-8587

[3] Assistant Professor, Department of CSE(AI & ML), Sri Venkatesa Perumal College of Engineering & Technology, Puttur, E-mail: bhaskark.mca@gmail.com , ORC-ID: https://orcid.org/0009-0000-3309-4240

Abstract: Predicting software defects is important for making software more reliable because it finds broken modules before they are deployed. This study shows a Hybrid Swarm Optimized Machine Learning Software Defect Prediction system that combines strong classifiers with metaheuristic feature selection. Artificial Bee Colony, Grasshopper Optimization Algorithm, GFGOA_ABC, and LGFGOA_ABC are some of the discriminative measures used in feature optimization. The Synthetic Minority Over sampling Technique is used to deal with class mismatch. XGBoost, Multi Layer Perceptron, and a Stacking Classifier that combines Random Forest, XGBoost, and Decision Tree models are used for classification. The GFGOA ABC Extension model works best on NASA MDP datasets CM1, KC1, KC4, PC3, and PC5. It gets 93.6 percent accuracy on CM1, 80.9 percent accuracy on KC1, 86.1 percent accuracy on KC4, 93.7 percent accuracy on PC3, and 84.0 percent accuracy on PC5. Explainable AI methods like LIME and SHAP make things clear by figuring out how features affect each other. For real-world use, the optimized model is put into action using a Flask-based web application that tells developers in real time which software modules are broken and which ones are not. This helps them make reliable quality assurance decisions for large development teams and continuous integration environments around the world.

"Index Terms - *Software Defect Prediction, Swarm Intelligence, Feature Selection, Hybrid Optimization, SMOTE, Stacking Classifier, Explainable AI, NASA MDP Datasets".*

1. INTRODUCTION

Improving the accuracy of hyperparameters is a key part of making machine learning models better at predicting the future. This is especially true in software defect prediction (SDP), where finding broken modules quickly is essential for making sure software works well. In safety-critical systems, even small flaws can lead to major problems, inconsistent operations, or catastrophic fails. This shows how important it is to find defects quickly and correctly [1, 7]. As more and more industries, like healthcare, defense, and finance, rely on software systems, using old software repositories to find bugs before they happen has become an important way to lower risks and support costs [2, 6].

Most of the time, probabilistic and simulation-based approaches to SDP fail to capture complex data dependencies. This is why machine learning and deep learning models are widely used for this job [8]. These models are very good at finding complex patterns and connections, which makes defect discovery faster and more reliable. Metaheuristic

algorithms (MAs) have also become popular as a way to solve problems in optimization, like hyperparameter setting and feature selection. MAs make global and local searches more efficient by combining exploration and exploitation processes [9]. This leads to better model training and better generalization. A lot of people use swarm intelligence-based algorithms because they can be scaled up or down, are flexible, and reach a conclusion quickly [3]. The No Free Lunch theorem, on the other hand, says that there is no one program that can always give the best results. This is why hybrid swarm-based optimization methods [5, 10] were created.

Recently done research shows that hybrid optimization methods can help make flaw prediction models more stable and accurate. SDP has come a long way thanks to methods like ensemble-based classification [4], reinforcement learning for feature selection [3], and hybrid sampling with optimization. Adding to what has already been done, this paper presents a Hybrid Swarm-Optimized Machine Learning Software Defect Prediction (HSoMLSDP) system. The main goal of this system is to combine swarm intelligence-driven feature optimization with strong classifiers to make predictions more accurate, make sure that classification is fair using SMOTE, and make things easier to understand using Explainable AI techniques such as LIME and SHAP.

2. RELATED WORK

In the past few years, a lot of study has been done on software defect prediction (SDP). Researchers have been trying to make SDP more accurate, reliable, and easy to understand by using different machine learning and optimization-based methods. A number of studies have stressed the importance of combining traditional classifiers with more advanced learning methods in order to deal with the uneven and complicated nature of software datasets. For example, Mehmood et al. [11] suggested a new framework based on machine learning that uses multiple classifiers and better preprocessing methods to make predictions more accurate. Their research showed that using the right feature engineering along with model optimization greatly improves the rate at which defects are found, especially when applied to large industrial datasets. In the same way, Daoud et al. [13] created a machine learning-based system that used a structured pipeline for preprocessing, feature selection, and classification. This showed that combined methods can be more accurate at predicting defects than single models.

Getting the best results from hyperparameters and feature selection has been another big area of research. Malhotra and Tyagi [12] came up with a way to tune parameters for SDP that is a mix of differential evolution and Tabu search. Their hybrid method made learning models work better by staying away from local optima and making convergence more stable. In the same vein, Balogun et al. [15] suggested a way for choosing wrapper features that is based on dynamic re-ranking strategies. Their system let features be reordered in a way that was adaptive during training, which improved the accuracy of classification across a wide range of datasets. Also, Mumtaz et al. [16] used an artificial immune network-based feature selection method to find important factors that affect the occurrence of defects. This bio-inspired method showed that immune network algorithms can successfully lower dimensionality while keeping important information about defects.

Class imbalance is another important study topic in SDP. Defective modules are usually in the minority compared to non-defective modules. To solve this

issue, Odejide et al. [14] did an empirical study that looked at various data selection methods. Their results showed that resampling techniques like oversampling and hybrid sampling help predict defects better by making the class distribution more even. This makes training and testing the model more fair. Alsheedi and Khan [18], who looked at supervised learning and ensemble methods, also talked about this problem. Their study showed that ensemble models do better on datasets that aren't balanced than single classifiers do because they combine several weak learners to get better generalization.

Over and over again, ensemble learning has been looked into in SDP study. Both Mehmood et al. [11] and Daoud et al. [13] talked about how ensemble methods can help make predictions more accurate. Jayanthi and Florence [19] looked into neural network-based classifiers that were trained on software measures. They found that combining multiple neural architectures made it easier to find modules that are more likely to have problems because they depend on other modules in nonlinear ways. Also, Cetiner and Sahingoz [17] compared different machine learning models and found that group methods that use decision trees, support vector machines, and neural networks work better than single models, especially on difficult test datasets.

Iqbal et al. [20], who did a full performance analysis of machine learning techniques on various NASA projects, showed that NASA datasets are still a standard way to test SDP approaches. Their research showed big differences in how well different algorithms worked, showing how important dataset features and hyperparameter settings are for predicting success. Another study by Alsheedi and Khan [18] showed that ensemble-based methods worked better across NASA datasets than standalone classifiers. The results support the idea that real-world software fault datasets are naturally noisy and unbalanced, so they need a mix of methods to be able to accurately predict what will happen.

3. MATERIALS AND METHODS

The proposed work presents HSoMLSDP, a Hybrid Swarm-Optimized Machine Learning framework for predicting software bugs that combines strong classifiers with advanced swarm intelligence algorithms. Artificial Bee Colony (ABC), Grasshopper Optimization Algorithm (GOA), and hybrid variants like LFGOA, GFGOA, GFGOA_ABC, and LGFGOA_ABC are used to find the best feature selection methods for finding the most important attributes for defect discovery. Along with baseline models, XGBoost and Multi-Layer Perceptron (MLP) are used for predictive modeling to improve accuracy and generalization. A Stacking Classifier that combines Random Forest, XGBoost, and Decision Tree is added to this framework to make it more robust by capturing different learning trends. Also, Explainable AI tools (LIME and SHAP) are built in to make things easier to understand by pointing out how features affect defect forecasts. This combined optimization and ensemble-based method improves the accuracy and openness of predictions. It does this by tackling the problems of feature selection, class imbalance, and model explainability in the prediction of software defects [22], [24], and [27].

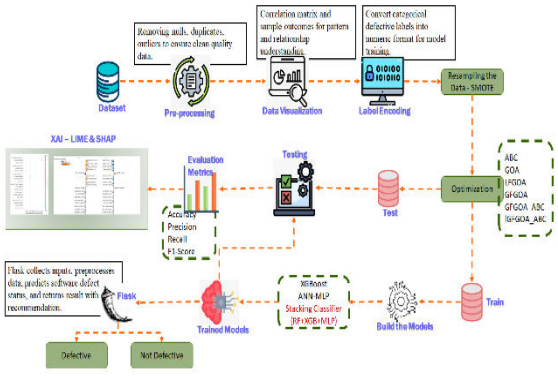


Fig.1 Proposed Architecture

Figure 1 shows a machine learning workflow for predicting software bugs. It starts with pre-processing the data and resampling it using SMOTE. After the data is optimized using different methods, such as GOA and ABC, it is split up so that XGBoost and Stacking Classifiers can build a model. Model testing, evaluation using standard metrics, and XAI interpretation are the last steps in the process. Flask is then used to release the software and label it as either broken or not.

i) Dataset Collection:

NASA's freely available software defect datasets were used to collect the data for this study. These datasets are commonly used in software reliability research to test prediction models [30]. These datasets give you measurements and labels for bugs in software modules from the past.

There are 327 software modules in the CM1 dataset. It has 41 characteristics that measure things like code complexity, size, and maintainability. These include LOC_BLANK, BRANCH_COUNT, CALL_PAIRS, CYCLOMATIC_COMPLEXITY, DECISION_COUNT, HALSTEAD metrics, and PATHOLOGICAL_COMPLEXITY. Each module is marked as either broken or not broken, which gives supervised learning a way to predict defects. Figure 2 shows the class distribution of CM1 modules, with most of the modules being good and only a few being bad. This shows that the dataset has an imbalance in the classes. This collection lets you test predictive models and optimization methods on real-life examples of software defects [30].

	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMATIC_COMPLEXITY	CYCLOMATIC_
0	6.0	9.0	2.0	1.0	0.0	16.0	5.0	
1	15.0	7.0	3.0	1.0	19.0	12.0	4.0	
2	27.0	9.0	1.0	4.0	22.0	16.0	5.0	
3	7.0	3.0	2.0	0.0	0.0	4.0	2.0	
4	51.0	25.0	13.0	0.0	14.0	48.0	13.0	

5 rows × 41 columns

Fig.2 CM1 Dataset

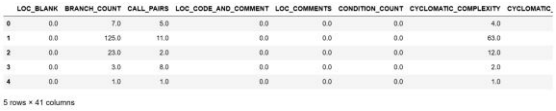
The KC1 dataset has 1,210 software modules and 22 attributes that show how structured, complicated, and easy to manage the code is. These attributes are LOC_BLANK, BRANCH_COUNT, LOC_CODE_AND_COMMENT, CYCLOMATIC_COMPLEXITY, DESIGN_COMPLEXITY, ESSENTIAL_COMPLEXITY, HALSTEAD metrics, and executable lines. Each module is marked as either broken or not broken, which allows for guided model training. The class distribution of KC1 is shown in Fig. 3. There are more non-defective modules than defective ones. This shows that the dataset is unequally distributed between classes and gives us a reasonable way to test prediction models [30].

	LOC_BLANK	BRANCH_COUNT	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CYCLOMATIC_COMPLEXITY	DESIGN_COMPLEXITY	ESSENTIAL_COMPLEXITY
0	0.0	1.0	0.0	0.0	1.0	1.0	1.0
1	0.0	1.0	0.0	0.0	1.0	1.0	1.0
2	2.0	1.0	0.0	0.0	1.0	1.0	1.0
3	2.0	1.0	0.0	0.0	1.0	1.0	1.0
4	1.0	1.0	0.0	0.0	1.0	1.0	1.0

5 rows × 22 columns

Fig.3 KC1 Dataset

The KC4 dataset has 119 software modules and 41 attributes that show how complicated, structured, and easy to manage the code is. These attributes include LOC_BLANK, BRANCH_COUNT, CALL_PAIRS, CYCLOMATIC_COMPLEXITY, DECISION_COUNT, HALSTEAD metrics, LOC_EXECUTABLE, and PATHOLOGICAL_COMPLEXITY. Each module is marked as either broken or not broken, which helps supervised learning find bugs. A balanced class representation is shown in Fig. 4, which shows how defective and non-defective modules are spread out. This makes it possible to test prediction models on small software datasets [30].

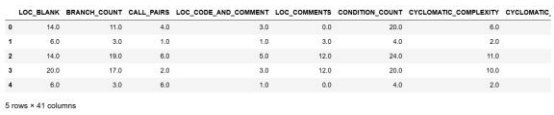


	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMATIC_COMPLEXITY	CYCLOMATIC_
0	0.0	7.0	5.0	0.0	0.0	0.0	4.0	
1	0.0	125.0	11.0	0.0	0.0	0.0	63.0	
2	0.0	23.0	2.0	0.0	0.0	0.0	12.0	
3	0.0	3.0	8.0	0.0	0.0	0.0	2.0	
4	0.0	1.0	1.0	0.0	0.0	0.0	1.0	

5 rows × 41 columns

Fig.4 KC4 Dataset

The PC3 dataset has 1,079 software modules and 41 attributes that show how structured, complicated, and easy to manage the code is. These attributes are LOC_BLANK, BRANCH_COUNT, CALL_PAIRS, LOC_CODE_AND_COMMENT, CYCLOMATIC_COMPLEXITY, DECISION_COUNT, HALSTEAD metrics, LOC_EXECUTABLE, and PATHOLOGICAL_COMPLEXITY. Supporting supervised learning, each lesson is marked as either broken or not broken. Figure 5 shows how defective and non-faulty modules are spread out. It shows that there is a class imbalance, with more non-defective modules than defective modules, which is important for evaluating the model and improving its performance [30].

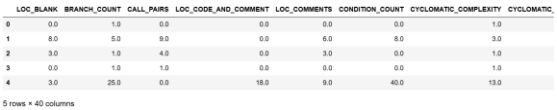


	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMATIC_COMPLEXITY	CYCLOMATIC_
0	14.0	11.0	4.0	3.0	0.0	20.0	6.0	
1	6.0	3.0	1.0	1.0	3.0	4.0	2.0	
2	14.0	19.0	6.0	5.0	12.0	24.0	11.0	
3	20.0	17.0	2.0	3.0	12.0	20.0	10.0	
4	6.0	3.0	6.0	1.0	0.0	4.0	2.0	

5 rows × 41 columns

Fig.5 PC3 Dataset

There are 1,830 software modules in the PC5 dataset. Each module is labeled as either defective or not defective, which helps supervised learning methods. The 40 attributes are LOC_BLANK, BRANCH_COUNT, CALL_PAIRS, LOC_CODE_AND_COMMENT, LOC_COMMENTS, CYCLOMATIC_COMPLEXITY, DECISION_COUNT, HALSTEAD metrics, LOC_EXECUTABLE, PATHOLOGICAL_COMPLEXITY, and NUMBER_OF_LINES. Figure 6 shows how defective and non-faulty modules are spread out. It shows that there are more non-defective modules than defective modules, which is important for fixing class imbalance in predictive modeling [30].



	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT	CYCLOMATIC_COMPLEXITY	CYCLOMATIC_
0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	
1	8.0	5.0	9.0	0.0	6.0	8.0	3.0	
2	3.0	1.0	4.0	0.0	3.0	0.0	1.0	
3	0.0	1.0	1.0	0.0	0.0	0.0	1.0	
4	3.0	25.0	0.0	18.0	9.0	40.0	13.0	

5 rows × 40 columns

Fig.6 PC5 Dataset

ii) Pre-Processing:

Pre-processing is an important part of data analysis because it gets rid of null values, duplicates, and outliers to make sure the quality of the information. When pre-processing is done right, it makes models more accurate, lowers noise, and makes future machine learning jobs more reliable.

a) Data Processing: Processing data means turning the attributes of a raw dataset into a structured shape that can be analyzed. In the beginning, null values and copies are removed to keep the data clean. Next, outliers that can mess up learning are found and removed. To make sure that everything is the same, continuous features can be normalized or scaled, and categorical features are changed so that they work with machine learning methods. This process also includes coming up with features and choosing which ones to use based on how well they fit the prediction job. Good data handling speeds up model convergence, simplifies computations, and makes sure that predictive models learn useful patterns from the dataset instead of just noise.

b) Data Visualization: Data visualization is an important part of analysis because it helps you see how the data is organized and how it shows patterns and connections. Methods like correlation matrices show the strength and direction of links between features, which helps find variables that are highly linked or duplicated. Visual plots, like histograms,

scatter plots, and boxplots, show how data is distributed, what the trends are, and where there might be problems. Sample results can also be seen to see how classes are distributed and whether there are any imbalances. This can help with preprocessing and resampling techniques. Visualization not only makes it easier to choose which features to use, but it also helps stakeholders understand what the dataset is like, which leads to better modeling and analysis choices.

c) Label Encoding: For model compatibility reasons, label encoding turns categorical labels into numeric values. This is especially useful for machine learning methods that need numerical inputs. In datasets for software defects, class names like "YES" or "NO" for broken modules are stored as integers, with 1 representing "YES" and 0 representing "NO." In this way, the category difference is kept while computing is made possible. Label encoding is important for supervised learning tasks because it makes sure that the algorithm can understand and improve results that are categorical. When you encode categorical data correctly, it can't be misinterpreted, and it can be handled the same way across all models and processes.

d) Data Resampling: Class mismatch can cause models to favor classes that are in the majority, but data resampling can fix this problem. Oversampling methods, like [26] SMOTE (Synthetic Minority Oversampling Technique), make fake samples for minority groups. This makes sure that there is an equal number of faulty and non-defective modules. Resampling makes classifiers work better by giving them enough examples to learn minority trends, stopping them from underfitting, and making their predictions more accurate. In software defect datasets, where bad modules usually only make up a small part of the dataset, this step is very important. When used with the right preparation and encoding, resampling makes sure that machine learning models can generalize well and catch rare events without being skewed by the majority class.

e) Feature Selection Using Optimization: Using optimization methods to pick out features aims to find the most important and relevant ones for predictive modeling while lowering the number of dimensions and making the computations easier. Several algorithms, including ABC (Artificial Bee Colony), GOA (Grasshopper Optimization Algorithm), LFGOA (Levy Flight GOA), GFGOA (Gaussian FGOA), GFGOA_ABC, and LGFGOA_ABC, test different sets of features over and over again using a fitness function and model success metrics. To avoid local optima and choose the best feature set, these metaheuristic methods strike a balance between exploring and exploiting. Focusing on informative characteristics, feature selection improves model accuracy, lowers overfitting, and makes it easier to understand, which leads to more efficient and accurate defect prediction models.

iii) Train & Test:

The train-test split is an important part of machine learning that checks how well the model works with data it hasn't seen before. The dataset is split into two parts. The training set, which is about 80% of the data, is used to learn patterns, connections, and how features interact with each other. The test set, which is the other 20% of the data, is used to make sure the model is correct. This split keeps the model from being tested on the same data it was trained on. This keeps the model from becoming too good at what it does and gives a more accurate picture of how it will do in the real world. For accurate and repeatable predictions, sorting the data correctly is very important.

iv) Algorithms:

Extreme Gradient Boosting, or XGBoost, is a more advanced machine learning method based on gradient boosting that mixes the outputs of several weak learners, usually decision trees, to make strong prediction models. A lot of people use it because it can handle big datasets with missing values quickly and accurately, and it can be scaled up or down. In defect prediction [21], XGBoost is used to figure out if a software section is likely to have bugs by studying how different features interact with each other. Its goal is to make predictions that are reliable, quick, and accurate while lowering overfitting through regularization. The method guarantees better generalization, which makes it perfect for situations where accurately classifying large amounts of data is necessary to improve system performance.

The final ensemble forecast model for XGBoost is shown below in the form of an equation.

$$\hat{y}_i = \sigma\left(\sum_{k=1}^{K} f_k(x_i)\right), f_k \in F \qquad (1)$$

An MLP is a special kind of artificial neural network that has an input layer, several hidden layers, and an output layer. It learns by changing weights using backpropagation and activation functions to find connections that don't follow a straight line. A lot of people use MLP to sort and guess things when simple models can't show how things depend on each other. In software defect prediction [23], MLP finds patterns in the input features to tell the difference between modules that are likely to have bugs and modules that are not likely to have bugs. Its goal is to model complex relationships in data, making accurate predictions through deeper learning, while also making sure that it can adapt to different problem types that are more complicated than standard linear models.

$$\hat{y} = f\left(W^L f\left(W^{L-1} \cdots f(W^1 X + b^1) + b^{(L-1)}\right) + b^L\right) \qquad (2)$$

A Stacking Classifier is a type of ensemble learning that uses more than one base learner to make predictions more accurate. [25] Random Forest, XGBoost, and Decision Tree are used as base classifiers in this case. The results of these classifiers are combined to make a meta-model. Stacking is used to take advantage of the good points of various methods while minimizing their bad points, which leads to better generalization. In defect detection, it brings together different points of view from different models to make the system more stable and improve classification accuracy. Its goal is to make predictions that are more stable, consistent, and accurate by lowering the mistakes that single models can make in data that is imbalanced or has a lot of dimensions.

The following is the Meta-Model Prediction equation:

$$\hat{y} = g(Y_{base}) = g\left(f_1(x), f_2(x), \ldots, f_m(x)\right) \qquad (3)$$

v) Integration of XAI and Flask Framework:

For example, LIME and SHAP are explainable AI methods that are used to figure out what the model predictions mean by finding important software metrics. This openness builds trust, helps with debugging, and lets developers understand and confirm choices about defect prediction that work well.

The Flask framework is used to make the improved model for predicting bugs work as a web-based app. It lets users work with it in real time, takes software metrics as input, and gives clear predictions about which modules are broken and which ones are not.

4. RESULTS AND DISCUSSIONS

Accuracy: How well a test can tell the difference between sick and healthy people is called its accuracy. To get an idea of how accurate a test is, we should figure out what percentage of cases are true positives and true negatives. In terms of math, this can be written as

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \quad (4)$$

Precision: Precision is the percentage of correctly classified cases or samples compared to those that were correctly classified as positives. So, here is the method to figure out the precision:

$$Precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \quad (5)$$

Recall: In machine learning, recall is a metric that shows how well a model can find all the important instances of a certain class. It shows how well a model captures instances of a certain class. It is calculated by dividing the number of correctly predicted positive observations by the total number of real positives.

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (6)$$

F1-Score: The F1 score is a way to rate the correctness of a machine learning model. It takes a model's accuracy and recall scores and adds them together. The accuracy metric counts how many times, across the whole dataset, a model made a correct guess.

$$F1\ Score = 2 * \frac{Recall\ \text{X}\ Precision}{Recall + Precision} * 100 \quad (7)$$

Table.1 Performance Evaluation – CM1 Dataset

ML Model	Accuracy	F1 Score	Recall	Precision	Running Time
ABC-XGBoost	0.889	0.889	0.889	0.894	0.335
ABC-ANN-MLP	0.526	0.394	0.526	0.972	0.078
ABC-Stacking Classifier	0.889	0.889	0.889	0.892	3.730
GOA-XGBoost	0.871	0.871	0.871	0.885	0.093
GOA-ANN-MLP	0.474	0.340	0.474	0.871	0.149
GOA-Stacking Classifier	0.912	0.912	0.912	0.916	2.407
GFGOA-XGBoost	0.877	0.877	0.877	0.878	0.220
GFGOA-ANN-MLP	0.503	0.474	0.503	0.617	0.148
GFGOA-Stacking Classifier	0.871	0.871	0.871	0.871	2.405
LFGOA-XGBoost	0.895	0.895	0.895	0.897	0.234
LFGOA-ANN-MLP	0.544	0.510	0.544	0.678	0.189
LFGOA-Stacking Classifier	0.906	0.906	0.906	0.907	2.509
GFGOA-ABC-XGBoost	0.918	0.918	0.918	0.921	0.229
GFGOA-ABC-ANN-MLP	0.398	0.397	0.398	0.399	0.215

GFGOA-ABC-Stacking Classifier	**0.936**	**0.936**	**0.936**	**0.937**	**2.503**
LFGOA-ABC-XGBoost	0.895	0.895	0.895	0.896	0.233
LFGOA-ABC-ANN-MLP	0.526	0.394	0.526	0.972	0.209
LFGOA-ABC-Stacking Classifier	0.912	0.912	0.912	0.913	2.576

Table.1 shows how well different optimization-based machine learning models did on the CM1 dataset by comparing their accuracy, F1 score, recall, precision, and time to run.

Fig.7 Comparison Graph – CM1 Dataset

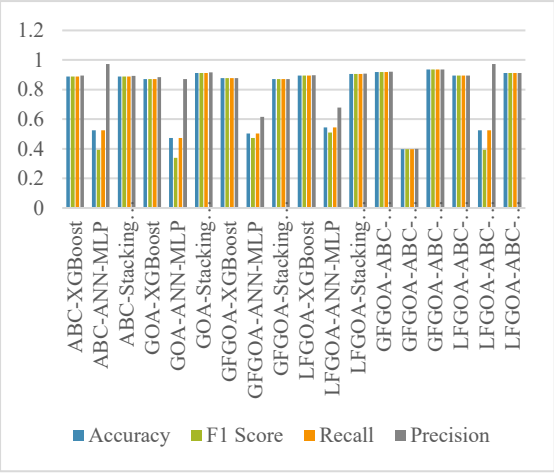


Figure 7 shows how well different mixed machine learning models do on four different metrics: Accuracy, F1 Score, Recall, and Precision.

Table.2 Performance Evaluation – KC1 Dataset

ML Model	**Accuracy**	**F1 Score**	**Recall**	**Precision**	**Running Time**
ABC-XGBoost	0.797	0.797	0.797	0.797	0.565
ABC-ANN-MLP	0.626	0.612	0.626	0.703	0.188
ABC-Stacking Classifier	0.797	0.797	0.797	0.799	3.964
GOA-XGBoost	**0.833**	**0.833**	**0.833**	**0.833**	**0.106**
GOA-ANN-MLP	0.664	0.658	0.664	0.695	0.425
GOA-Stacking Classifier	0.822	0.821	0.822	0.822	4.114
GFGOA-XGBoost	0.803	0.803	0.803	0.803	0.300
GFGOA-ANN-MLP	0.667	0.666	0.667	0.678	0.729
GFGOA-Stacking Classifier	0.796	0.795	0.796	0.797	4.120
LFGOA-XGBoost	0.812	0.812	0.812	0.812	0.305
LFGOA-ANN-MLP	0.628	0.620	0.628	0.672	0.745
LFGOA-Stacking Classifier	0.812	0.812	0.812	0.815	3.115
GFGOA-ABC-XGBoost	0.820	0.819	0.820	0.822	0.303

GFGOA-ABC-ANN-MLP	0.574	0.508	0.574	0.843	0.254
GFGOA-ABC-Stacking Classifier	0.809	0.808	0.809	0.816	3.107
LFGOA-ABC-XGBoost	0.807	0.806	0.807	0.817	0.306
LFGOA-ABC-ANN-MLP	0.578	0.538	0.578	0.753	0.220
LFGOA-ABC-Stacking Classifier	0.790	0.787	0.790	0.814	3.124

Table.2 shows how well different optimization-based machine learning models did on the KC1 dataset, focusing on accuracy, F1 score, recall, precision, and running time.

Fig.8 Comparison Graph – KC1 Dataset

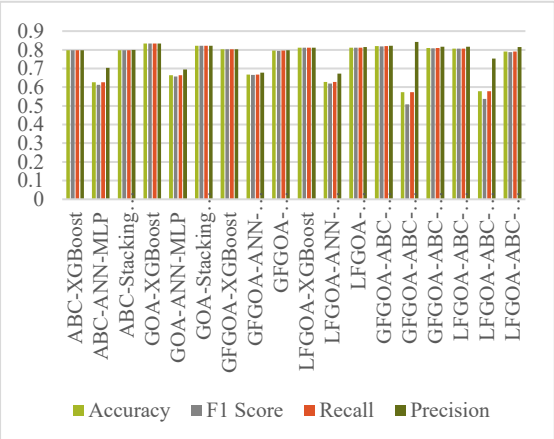


Figure 8 shows how well different hybrid machine learning models did on the KC1 dataset, as judged by Accuracy, F1 Score, Recall, and Precision.

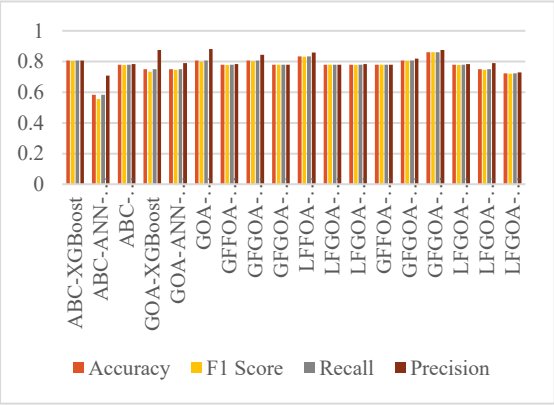
Table.3 Performance Evaluation – KC4 Dataset

ML Model	Accuracy	F1 Score	Recall	Precision	Running Time
ABC-XGBoost	0.806	0.805	0.806	0.807	0.265
ABC-ANN-MLP	0.583	0.556	0.583	0.708	0.016
ABC-Stacking Classifier	0.778	0.777	0.778	0.784	1.921
GOA-XGBoost	0.750	0.733	0.750	0.875	0.060
GOA-ANN-MLP	0.750	0.745	0.750	0.789	0.246
GOA-Stacking Classifier	0.806	0.798	0.806	0.881	2.034
GFFOA-XGBoost	0.778	0.777	0.778	0.784	0.152
GFGOA-ANN-MLP	0.806	0.802	0.806	0.844	0.020
GFGOA-Stacking Classifier	0.778	0.778	0.778	0.778	1.953
LFFOA-XGBoost	0.833	0.831	0.833	0.858	0.163
LFGOA-ANN-MLP	0.778	0.778	0.778	0.778	0.254
LFGOA-Stacking Classifier	0.778	0.777	0.778	0.784	1.949
GFFOA-ABC-	0.778	0.778	0.778	0.778	0.156

XGBoost					
GFGOA-ABC-ANN-MLP	0.806	0.804	0.806	0.819	0.264
GFGOA-ABC-Stacking Classifier	**0.861**	**0.860**	**0.861**	**0.875**	**1.942**
LFGOA-ABC-XGBoost	0.778	0.777	0.778	0.784	0.150
LFGOA-ABC-ANN-MLP	0.750	0.745	0.750	0.789	0.264
LFGOA-ABC-Stacking Classifier	0.722	0.721	0.722	0.728	1.952

Accuracy, F1 Score, Recall, and Precision were used to rate how well different mixed machine learning models did on the KC1 dataset. The results are shown in Figure 8.

Fig.9 Comparison Graph – KC4 dataset



Accuracy, F1 Score, Recall, and Precision were used to measure how well different mixed machine learning models worked on the KC4 dataset (Fig. 9).

Table.4 Performance Evaluation – PC3 Dataset

ML Model	Accuracy	F1 Score	Recall	Precision	Running Time
ABC-XGBoost	0.922	0.922	0.922	0.923	0.432
ABC-ANN-MLP	0.520	0.499	0.520	0.604	0.379
ABC-Stacking Classifier	0.922	0.922	0.922	0.923	4.595
GOA-XGBoost	0.887	0.887	0.887	0.892	0.114
GOA-ANN-MLP	0.739	0.728	0.739	0.822	0.559
GOA-Stacking Classifier	0.914	0.914	0.914	0.914	3.396
GFFOA-XGBoost	0.907	0.906	0.907	0.910	0.302
GFGOA-ANN-MLP	0.753	0.743	0.753	0.829	0.943
GFGOA-Stacking Classifier	0.924	0.924	0.924	0.924	3.349
LFFOA-XGBoost	0.901	0.901	0.901	0.901	0.303
LFGOA-ANN-MLP	0.656	0.643	0.656	0.732	0.853
LFGOA-Stacking Classifier	0.914	0.914	0.914	0.914	3.108
GFFOA-ABC-	0.933	0.933	0.933	0.935	0.312

XGBoost					
GFGOA-ABC-ANN-MLP	0.501	0.334	0.501	1.000	0.199
GFGOA-ABC-Stacking Classifier	**0.937**	**0.937**	**0.937**	**0.937**	**3.168**
LFGOA-ABC-XGBoost	0.921	0.921	0.921	0.921	0.307
LFGOA-ABC-ANN-MLP	0.501	0.337	0.501	0.998	0.448
LFGOA-ABC-Stacking Classifier	0.917	0.917	0.917	0.917	3.172

Table.4 shows how well optimization-based machine learning models did on the PC3 dataset by showing their accuracy, F1 score, recall, precision, and running time.

Fig.10 Comparison Graph – PC3 Dataset

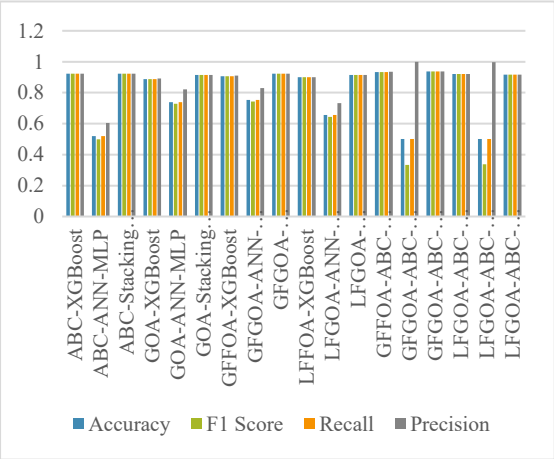


Figure 10 shows how well different hybrid machine learning models did on the PC3 dataset, as judged by Accuracy, F1 Score, Recall, and Precision.

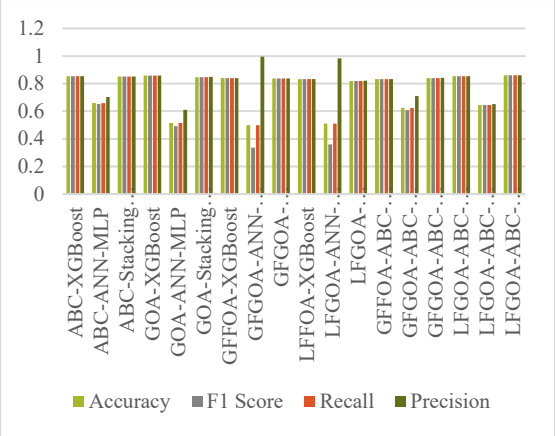
Table.5 Performance Evaluation – PC5 Dataset

ML Model	Accuracy	F1 Score	Recall	Precision	Running Time
ABC-XGBoost	0.853	0.853	0.853	0.853	0.608
ABC-ANN-MLP	0.660	0.652	0.660	0.703	0.740
ABC-Stacking Classifier	0.851	0.851	0.851	0.852	5.538
GOA-XGBoost	0.858	0.858	0.858	0.858	0.120
GOA-ANN-MLP	0.516	0.492	0.516	0.610	0.592
GOA-Stacking Classifier	0.847	0.846	0.847	0.848	4.181
GFFOA-XGBoost	0.839	0.839	0.839	0.839	0.327
GFGOA-ANN-MLP	0.500	0.336	0.500	0.995	0.629
GFGOA-Stacking Classifier	0.838	0.838	0.838	0.838	4.166
LFFOA-XGBoost	0.832	0.832	0.832	0.832	0.326
LFGOA-ANN-MLP	0.511	0.360	0.511	0.984	0.545
LFGOA-Stacking	0.819	0.819	0.819	0.821	3.295

Classifier					
GFFOA-ABC-XGBoost	0.834	0.834	0.834	0.834	0.329
GFGOA-ABC-ANN-MLP	0.625	0.608	0.625	0.710	0.682
GFGOA-ABC-Stacking Classifier	0.840	0.840	0.840	0.843	3.325
LFGOA-ABC-XGBoost	0.853	0.853	0.853	0.854	0.338
LFGOA-ABC-ANN-MLP	0.646	0.645	0.646	0.651	0.721
LFGOA-ABC-Stacking Classifier	**0.860**	**0.860**	**0.860**	**0.861**	**3.342**

Table.5 shows how well different optimization-based machine learning models did on the PC5 dataset, focusing on running time, accuracy, F1 score, recall, and precision.

Fig.11 Comparison Graph – PC5 Dataset



Accuracy, F1 Score, Recall, and Precision were used to measure how well different mixed machine learning models worked on the PC5 dataset (Fig. 11).

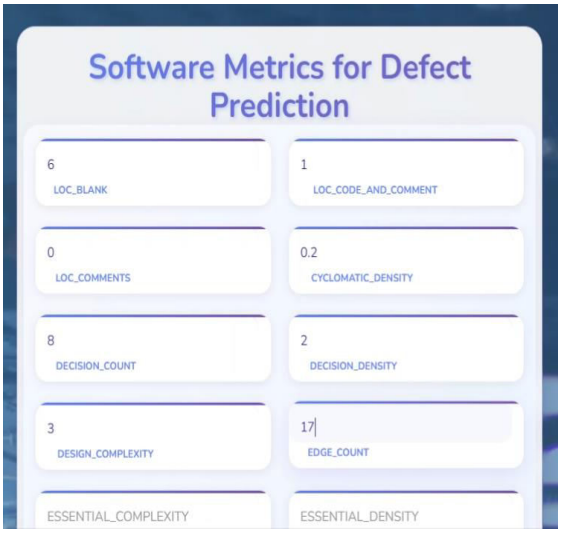


Fig.12 Upload Input Data – CM1 Dataset

In order to predict software defects, Fig. 12 shows a user interface for entering software measurements from the CM1 dataset.

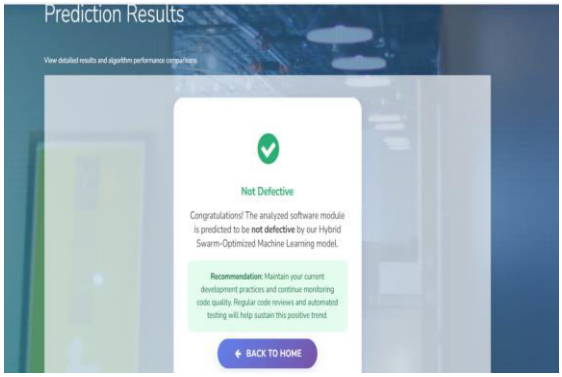


Fig.13 Predict Result

The model's forecast result for the given input data is shown in Fig.13, which means it has been labeled as Not-Defective.

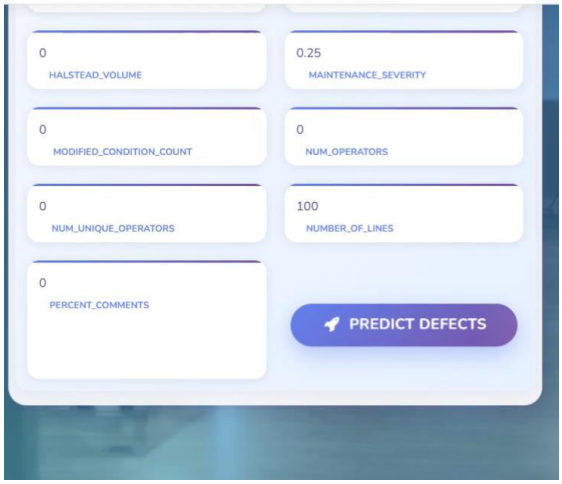


Fig.14 Upload Input Data – KC4 Dataset

The uploaded input values in Fig.14 are used by the model to create and correctly predict the outcome.

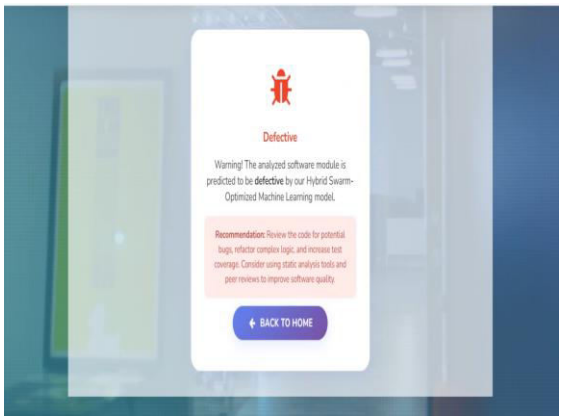


Fig.15 Final Outcome

The expected result for the given input values is shown in Fig.15, which means that the software module is considered to be broken.

5. CONCLUSION

The suggested HSoMLSDP approach shows that hybrid swarm optimized machine learning makes software defect prediction much more accurate and reliable. The tests show that using GFGOA ABC optimized features along with a Stacking Classifier gives better results across all NASA MDP datasets, with 93.6 percent accuracy on CM1, 80.9 percent accuracy on KC1, 86.1 percent accuracy on KC4, 93.7 percent accuracy on PC3, and 84.0 percent accuracy on PC5. These results show that it is possible to make strong generalizations while still keeping computational performance good. Explainable AI techniques, like LIME and SHAP, improve openness by making it clear which software metrics are important. This lets developers believe and confirm predictions. The optimized model is put into use through a lightweight Flask-based web app that allows real-time inference so that it can be used in the real world. Software units are marked as either broken or not broken using this interface, which helps quality assurance teams reduce risks early on. Overall, the framework creates an easy-to-understand, scalable, and accurate decision support system that lowers the need for testing, boosts reliability, and helps with proactive defect management in a wide range of software development environments. It also provides consistent results that can be used for continuous integration pipelines and long-term maintenance planning across enterprise projects.

In the future, researchers will add more metaheuristic optimization techniques to the suggested hybrid swarm-optimized machine

learning framework and look into deep ensemble models to make predictions that are more reliable. To get even better results, you can use more complicated neural architectures, like attention-driven and graph-based networks, to understand how features depend on each other. The tool will be better at generalization if it can be used to test on bigger and more varied sets of defects. Also, attempts will be made to lower the cost of computing by using lightweight optimization strategies that keep the accuracy of the predictions high. Building adaptive models that are always learning from new software data is another important area. These models can make predictions in real time and find bugs early in environments that are always changing, which improves the quality and reliability of software.

REFERENCES

[1] Dong, X., Wang, J., & Liang, Y. (2025). A Novel Ensemble Classifier Selection Method for Software Defect Prediction. *IEEE Access*.

[2] G, Viswanath., N, Madhvik., K, Bhaskar., K, Supriya. (2024). Machine-Learning-Based Cloud Intrusion Detection. International Journal of Mechanical Engineering Research and Technology, 16(9), 38-52.

[3] Hesamolhokama, M., Shafiee, A., Ahmaditeshnizi, M., Fazli, M., & Habibi, J. (2024). SDPERL: A Framework for Software Defect Prediction Using Ensemble Feature Extraction and Reinforcement Learning. arXiv preprint arXiv:2412.07927.

[4] Bashir, A. T., Balogun, A. O., Adigun, M. O., Ajagbe, S. A., Capretz, L. F., Awotunde, J. B., & Mojeed, H. A. (2024, April). Cascade Generalization-Based Classifiers for Software Defect Prediction. In *Computer Science On-line Conference* (pp. 22-42). Cham: Springer Nature Switzerland.

[5] Taskeen, A., Khan, S. U. R., & Mashkoor, A. (2024). An adaptive synthetic sampling and batch generation-oriented hybrid approach for addressing class imbalance problem in software defect prediction. Soft Computing, 28(23), 13595-13614.

[6] M. K. Suryadi, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, ‘‘Comparative study of various hyperparameter tuning on random forest classification with SMOTE and feature selection using genetic algorithmin software defect prediction,’’ J. Electron., Electromedical Eng., Med. Infor mat., vol. 6, no. 2, pp. 137–147, Mar. 2024.

[7] D. N. Sharma and D. K. Yadav, ‘‘Machine learning based approach for software defect prediction using hyperparameter,’’ Tech. Rep., 2024.

[8] M. Ali, T. Mazhar, Y. Arif, S. Al-Otaibi, Y. Yasin Ghadi, T. Shahzad, M. Amir Khan, and H. Hamam, ‘‘Software defect prediction using an intelligent ensemble-based model,’’ IEEE Access, vol. 12, pp. 20376–20395, 2024.

[9] S. P. Shankar and S. S. Chaudhari, ‘‘Analysis of bio inspired based hybrid learning model for software defect prediction,’’ Social Netw. Comput. Sci., vol. 5, no. 7, p. 825, Aug. 2024.

[10] Viswanath, G., Abirami, V., & Prathyusha, G. (2024). Hybrid Feature Extraction With Machine Learning To Identify Network Attacks. International Journal Of HRM And Organizational Behavior, 12(3), 217-228.

[11] I. Mehmood, S. Shahid, H. Hussain, I. Khan, S. Ahmad, S. Rahman, N. Ullah, and S. Huda, ‘‘A novel approach to improve software defect

prediction accuracy using machine learning,'' IEEE Access, vol. 11, pp. 63579–63597, 2023.

[12] R. Malhotra and A. Tyagi, ''Hybrid differential evolution and Tabu search for parameter tuning in software defect,'' in Proc. IEEE 7th Int. Conf. Converg. Technol. (I2CT), Apr. 2022, pp. 1–7.

[13] M. Sh. Daoud, S. Aftab, M. Ahmad, M. Adnan Khan, A. Iqbal, S. Abbas, M. Iqbal, and B. Ihnaini, ''Machine learning empowered software defect prediction system,'' Intell. Autom. Soft Comput., vol. 31, no. 2, pp. 1287–1300, 2022.

[14] B. J. Odejide, A. O. Bajeh, A. O. Balogun, Z. O. Alanamu, K. S. Adewole, A. G. Akintola, S. A. Salihu, F. E. Usman-Hamza, and H. A. Mojeed, ''An empirical study on data sampling methods in addressing class imbal ance problem in software defect prediction,'' in Proc. Comput. Sci. On-line Conf. Cham, Switzerland: Springer, Jan. 2022, pp. 594–610.

[15] Kumar, K., Udaya Suriya Rajkumar, D., Viswanath, G., & Mahalakshmi, J. (2024). A Hybrid Particle Swarm Optimization and C4.5 for Network Intrusion Detection and Prevention System. International Journal of Computing, 23(1), 109-115. https://doi.org/10.47839/ijc.23.1.3442

[16] B. Mumtaz, S. Kanwal, S. Alamri, and F. Khan, ''Feature selection using artificial immune network: An approach for software defect prediction,'' Intell. Autom. Soft Comput., vol. 29, no. 3, pp. 669–684, 2021.

[17] M. Cetiner and O. K. Sahingoz, ''A comparative analysis for machine learning based software defect prediction systems,'' in Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT), Jul. 2020, pp. 1–7.

[18] A.Alsaeedi andM.Z.Khan,''Software defect prediction using supervised machine learning and ensemble techniques: A comparative study,'' J. Softw. Eng. Appl., vol. 12, no. 5, pp. 85–100, 2019.

[19] R. Jayanthi and L. Florence, ''Software defect prediction techniques using metrics based on neural network classifier,'' Cluster Comput., vol. 22, no. S1, pp. 77–88, Jan. 2019.

[20] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana, M. Ahmad, and A. Husen, ''Performance analysis of machine learning techniques on software defect prediction using NASA datasets,'' Int. J. Adv. Comput. Sci. Appl., vol. 10, no. 5, pp. 1–9, 2019.

[21] B. Lemon, ''The effect of locality based learning on software defect prediction,'' West Virginia Univ., Morgantown, WV, USA, Tech. Rep., 2010.

[22] Malhotra, R., Chawla, S., & Sharma, A. (2025). Software defect prediction based on multi-filter wrapper feature selection and deep neural network with attention mechanism. *Neural Computing and Applications*, 1-28.

[23] Chen, L. F., Zhang, S. P., Qin, Y. Y., Cao, K. X., Du, T., & Dai, Q. (2025). Software defect prediction based on support vector machine optimized by reverse differential chimp optimization algorithm. *International Journal of Data Science and Analytics*, 1-26.

[24] Keya, A. J., Khandaker, N., Hasan, M. J., Jahan, H., & Akhtaruzzaman, M. (2024, December). Leveraging CatBoost and XAI for Enhanced Software Defect Prediction. In *2024 27th International Conference on Computer and Information Technology (ICCIT)* (pp. 2998-3003). IEEE.

[25] Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Ghadi, Y. Y., & Khan, M. A. (2024). Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. *PeerJ Computer Science*, *10*, e1860.

[26] Lakshmi, J. M., Prasad, K. K., & Viswanath, G. (2025). Proactive Security in Multi-Cloud Environments: A Blockchain Integrated Real-Time Anomaly Detection and Mitigation Framework. Cuestiones De Fisioterapia, 54(2), 392-417.

[27] Lakshmi, P. J., Krishna, T. S., Kumar, N. B., Rao, D. N. M., Hosseinpour, A., & Lenin, N. C. (2025). Improving Software Fault Prediction with a Hybrid DE-WOA Optimizer and ANFIS-Enhanced Ensemble Learning. *IEEE Access*.

[28] Malhotra, R., & Khan, K. (2024, September). Hyperparameter Optimization in Deep Learning for Improved Software Defect Prediction: A Stacked-Ensemble Approach. In *Congress on Intelligent Systems* (pp. 249-262). Singapore: Springer Nature Singapore.

[29] Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y. Y., Shahzad, T., ... & Hamam, H. (2024). Software defect prediction using an intelligent ensemble-based model. *IEEe Access*, *12*, 20376-20395.

[30] Siddiqui, T., & Mustaqeem, M. (2023). Performance evaluation of software defect prediction with NASA dataset using machine learning techniques. *International Journal of Information Technology*, *15*(8), 4131-4139.