Computer & Robotics

Layered Defect Prediction Model for Software Product Lines Using Feature Fusion and Ensemble Classification

Mehdi Habibzadeh khameneh[a,b], Akbar Nabiollahi-Najafabadi[a,b,*], Reza Tavoli[c], Hamid Rastegari[a,b]

[a] *Faculty of Computer Engineering, Najafabad branch, Islamic Azad University, Najafabad, Iran*

[b] *Big Data Research Center, Najafabad branch, Islamic Azad University, Najafabad, Iran*

[c] *Department of Computer Engineering, Chalous Branch, Islamic Azad University, Chalous, Iran*

Received 22 April 2025, Accepted 29 June 2025.

Abstract

Software defect prediction in Software Product Lines (SPLs) presents significant challenges due to high-dimensional feature spaces and intricate feature dependencies. We introduce a novel three-layered framework that integrates (1) optimal feature selection using six metaheuristic algorithms (HHO, ACO, GWO, GA, PSO, WOA), (2) feature fusion to create a unified and enriched feature representation, and (3) a stacked ensemble classifier comprising KNN, Decision Tree, Naive Bayes, and XGBoost. Unlike prior studies that rely on single optimization or classification models, our approach combines multiple optimizers and learners in a synergistic pipeline, enhancing generalization and robustness. The proposed method was rigorously evaluated on benchmark datasets from NASA (CM1, JM1, KC1, PC1) and the Linux Variable Analysis Tools (LVAT) repository (LTS1, LTM2, LTL3, LTV4). It achieved 98.62%, 97.81%, 98.59%, and 98.71% accuracy rates on the NASA datasets and 95.1%, 94.2%, 97.3%, and 99.4% on the LVAT datasets, respectively. These results demonstrate that the proposed approach consistently outperforms existing methods across diverse SPL scenarios.

Keywords: Software Product lines, Defect Prediction, Metaheuristic Algorithms, Ensemble Learning, Feature Fusion

1.Introduction

Software Product Lines (SPLs) aim to reuse software assets systematically to derive diverse product variants. However, managing the inherent variability and maintaining product consistency poses significant challenges, especially during feature selection and configuration [1]. In software product line engineering, Feature Models (FMs) are commonly employed to represent the variability and commonality within a family of software products [2]. A tertiary study categorizes approaches to managing variability in SPLs into three main types: Feature Models (FM), Orthogonal Variability Models (OVM), and Decision Models (DM). Among these, this study focuses on FMs, the most recognized and widely used method for variability modeling [3]. FMs incorporate various constructs, such as Optional and mandatory features, to Determine whether a feature must always be present in a product. Alternative ("xor") and selection ("or") groups Govern exclusive feature selections and permissible feature combinations. Constraints (requires/excludes): Specify dependencies or incompatibilities between features [4]. Without automated support, the feature selection process is often inefficient and error-prone. It involves satisfying multiple, sometimes conflicting, objectives, such as aligning with user preferences, minimizing product costs, and adhering to technical feasibility constraints. This challenge is compounded in feature spaces with thousands of features, where human intuition alone is inadequate to identify

*Corresponding Author. Email: a.nabi@pco.iaun.ac.ir

optimal or near-optimal software product configurations. Automated techniques for feature selection have been developed [5] to address these complexities. Recent advances [6, 7] have similarly explored the use of metaheuristics in combination with deep learning, highlighting the growing trend toward hybrid approaches and reinforcing the rationale. In this paper, the proposed approach comprises three layers: The first layer preprocesses the Data set and utilizes meta-heuristic algorithms to extract an optimal subset of features. In the second layer, these selected features are fused to form an integrated feature set, which is then forwarded to the third layer. In this final layer, stacking-based ensemble learning techniques are employed to enhance the performance of product defect detection. The proposed methodology was validated through simulation experiments on eight Data sets: four from NASA and four from the Linux Variable Analysis Tools (LVAT). This study introduces an enhanced architecture that integrates metaheuristic-based feature selection with stacking ensemble classification to achieve robust and scalable defect prediction in SPLs.

The main contributions of this article are as follows:

• Proposing a novel three-layer architecture for software defect prediction in SPLs that integrates metaheuristic-based feature selection, feature fusion, and ensemble learning.

• Employing six distinct metaheuristic algorithms (HHO, ACO, GWO, GA, PSO, WOA) to enhance feature diversity and optimize the selection process.

• Designing a stacked ensemble classifier (KNN, DT, NB, XGBoost) that improves prediction robustness and generalization.

• Validating the proposed model on both NASA and LVAT datasets and demonstrating superior performance across standard metrics compared to state-of-the-art methods.

The article continues as follows: Sect. 2 Explains the fundamental concepts and background of the research. The proposed system's theoretical background is explained in section 3. Section 4 presents the proposed methodology. The experimental setup, Data sets, evaluation criteria, research results, and comparisons with previous works in Sect. 5 discusses the performance measures and the environmental setup. Section 6 The paper concludes and offers suggestions for future research directions.

2.Related works

Over recent years, diverse approaches have been proposed to enhance software defect detection, particularly in feature selection and classification optimization. For instance, Ref. [8] ensemble learning techniques boosting, bagging, and stacking were applied for code smell detection, leading to improvements of up to 21.43% in accuracy, 53.24% in AUC, and 76.06% in F-measure. Additionally, the combination of XGBoost and stacking was used to address the challenges of high-dimensional defect data, achieving prediction accuracies above 90% across various Data sets [9]. Furthermore, feature identification in software product lines was explored using LSI, PCA, and TF-IDF, with LSI outperforming the others at 68.6% accuracy [10]. Reference [11], evaluated three deep learning models (LSTM, BiLSTM, and RBFN), where LSTM and BiLSTM achieved prediction accuracies of 93.53% and 93.75%, respectively, while RBFN was faster but less accurate. A hybrid version, combining the Non-Dominated Sorting Genetic Algorithm (NSGA-II) with the k-nearest Neighbor (k-NN) classifier, demonstrated superior performance based on the Matthews correlation coefficient (MCC) [12]. Reference [13], a hybrid algorithm integrating Golden Jackal and Grey Wolf optimization strategies was proposed for feature selection, which improved accuracy and reduced runtime when coupled with a Random Forest classifier. In [14], metaheuristic algorithms, including GA, PSO, and GWO, were used for anomaly detection in financial Data sets, combined with classifiers such as SVM. The results showed significant improvements in detection performance, although the efficiency of the approach requires further validation across diverse domains. Furthermore, Ref. [15], an advanced hybrid of WOA, PSO, and Firefly Algorithm (FA), was combined with Deep Q-learning, improving multiple evaluation metrics, including up to 9.5% in precision and 8.5% in accuracy. Similarly, In [16], GA and GWO were employed for anomaly detection in financial data, reaching a maximum accuracy of 75.83% using GWO. An ensemble-based software defect prediction model integrating RF, SVM, Naive Bayes, and ANN was presented in [17], demonstrating improved accuracy compared to other individual models. A hybrid method combining the Grasshopper Optimization Algorithm (GOA), Genetic Algorithm (GA), and Random Forest was proposed in [18] for

intrusion detection systems, achieving accuracies of 98%, 99%, and 92% across multiple Data sets. Reference [19], a five-step framework using ensemble machine learning and improved feature selection attained up to 95.1% accuracy. In [20], a comprehensive evaluation of various machine learning algorithms, including logistic regression, SVM, AdaBoost, Naive Bayes, decision trees, and MLP, was conducted across multiple open-source software Data sets. The study introduced additional metrics to enhance software error prediction, showing moderate performance improvements with high computational demands. Reference [21], the LTHFFA algorithm was proposed, incorporating BAVSSA, KPCA, and LLE for enhanced software error prediction and effective dimensionality reduction. In [22], a hybrid CNN–MLP classifier is employed. The convolutional neural network (CNN) processes semantic features extracted from project abstract syntax trees (ASTs) using Word2Vec, while traditional features obtained from the dataset repository are processed by a multilayer perceptron (MLP). The outputs of both CNN and MLP are then fused and passed to a fully connected layer for defect prediction. In [23] various data mining classification methods are employed, including neural networks, K-nearest neighbor, support vector machine, logistic regression, decision tree, and random forest. This study investigates the combined impact of data balancing, acceleration algorithms, and high-accuracy classifiers—particularly neural networks. The main contribution of the paper is the introduction of a speed accuracy trade-off framework for hybrid classifiers aimed at addressing real-world defect prediction challenges. Recent studies, such as Ref. [6] employ multiple metaheuristic algorithms to optimize machine learning models for software defect prediction. Similarly, Ref. [7] integrates deep learning techniques with a fish migration optimization algorithm to enhance predictive performance. In Ref. [24] the authors propose a hybrid approach that combines Harris Hawks Optimization (HHO) with stacking-based ensemble learning to improve the accuracy and robustness of software defect prediction. These prior works demonstrate the ongoing evolution of software defect prediction methodologies and highlight the effectiveness of combining metaheuristic optimization with ensemble learning. Building on this foundation, the current study proposes a novel multi-layered architecture that integrates metaheuristic-based feature selection, feature fusion, and a stacking ensemble learning model to further enhance defect detection accuracy in SPL environments.

3.Proposed Approach

Software Product Line (SPL) represents a significant methodology in software engineering, focusing on systematically developing software products that share common features. Feature Model (FM) captures the commonalities and variabilities among a set of related software products within a specific domain. Product development can be successfully achieved by incorporating a few key, well-chosen features. However, selecting these features becomes more complex when dependencies (AND, OR, Exclusive, and Require) are involved. Product inconsistencies occur when the selection of configured features violates configuration constraints, leading to an inconsistent product. Inconsistency is a critical defect that undermines the advantages of Software Product Lines (SPL). Meta-heuristic algorithms can help detect and resolve these inconsistencies by selecting optima or near optima features faster and more accurately. This study proposes a three-layer architecture that combines meta-heuristic algorithms with machine learning to product defect detection in software production lines. Firstly, a feature extraction using six metaheuristic algorithms (HHO, ACO, GWO, GA, PSO, WOA). The goal is to identify the vector containing the most relevant features, optimizing the objective function for feature selection. In the second phase, a feature fusion that aggregates selected subsets into a unified feature vector. In the third phase, we use a stacked ensemble learning approach combining KNN, DT, and NB with XGBoost as the meta-classifier to model and classify software product line products. This method combines multiple machine learning models to enhance predictive accuracy. Through this process, inconsistencies in the software production line can be detected and addressed, improving the overall robustness and quality of the product line. A Software Product Line (SPL) comprises a set of related software products that share a common core while differing in specific features. Effective feature selection in SPLs aims to align with user priorities, reduce development costs, ensure technical feasibility in large feature spaces, minimize production time, and improve overall quality and efficiency. Software defects often emerge from configuration rule violations that cause product incompatibilities.

To address this, meta-heuristic algorithms are employed for dimensionality reduction, effectively facilitating defect detection in high-dimensional feature spaces. This study presents a three-layer architecture designed to predict software defects in SPLs more accurately and efficiently. The core idea is to integrate advanced feature selection techniques with ensemble learning.

Overview of the Proposed Approach:

1.Preprocessing: Initial data preparation is performed on the SPL Data set to ensure consistency and readiness for analysis.

2.First Layer: Feature Extraction: Six well-known metaheuristic algorithms (HHO, ACO, GWO, GA, PSO, and WOA) are independently applied to the preprocessed Data set to select relevant features. These methods reduce dimensionality while preserving important predictive attributes.

3.Second Layer: Feature Fusion: The selected feature subsets are combined into a unified, enriched feature vector. This fusion enhances representation by aggregating both shared and complementary features from each algorithm.

4.Third Layer: Group Classification: A stacking-based ensemble model is implemented using KNN, Decision Tree, and Naive Bayes as base learners, with XGBoost serving as the meta-classifier. This configuration aims to improve generalization and classification accuracy.

5. Model Evaluation: The model's performance is assessed using a confusion matrix and standard metrics, including accuracy, recall, precision, F1-score, classification error (E), and a custom fitness function. The results are compared with existing methods to highlight improvements.

This layered approach enables fast and accurate detection of software defects, leveraging the strengths of meta-heuristic search and deep neural representations. The workflow is visually summarized in Figure 1.

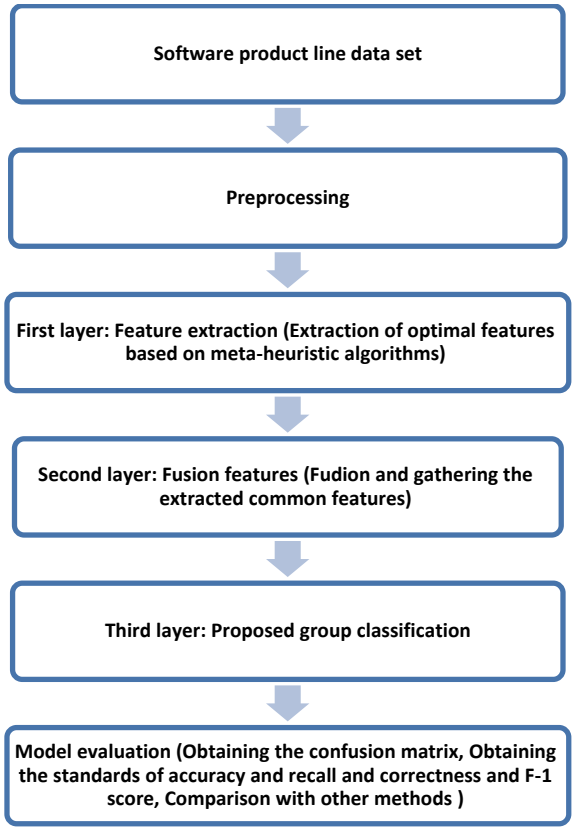

Fig. 1.General steps of the proposed research process

4.Methodology

The proposed methodology is structured into three core layers: feature extraction, feature fusion, and ensemble classification. This architecture is designed to enhance the accuracy and robustness of software defect prediction in Software Product Lines (SPLs), particularly in high-dimensional feature spaces with complex dependencies. Each layer plays a specific role in processing and learning from the data to improve the final classification output. The fitness function used in this study is adapted from general formulations commonly used in the literature for multi-objective feature selection. While it is not directly copied from any one source, it reflects well-established design principles, and we have cited representative examples accordingly.

4.1.Preprocessing

The first step in the pipeline is to ensure the Data set is clean, normalized, and ready for modeling. Preprocessing includes missing value imputation, noise filtering, and normalization. Noise removal was performed by identifying and correcting outliers using statistical methods, such as Z-scores.

Normalization is particularly important in this context since the input features often differ in scale and range, potentially biasing the results of certain algorithms. By bringing all features to a common scale, the learning models can treat each feature equally. Initial filtering techniques are also applied to eliminate highly correlated or irrelevant features before invoking optimization-based selection. This helps reduce computational load and guides the metaheuristic algorithms more efficiently toward meaningful feature subsets.

4.2.Extracting Optimal Features with Meta-heuristic Algorithms

Our approach formulates the feature selection process as a binary combinatorial optimization problem. The objective is to identify an optimal subset of features from a high-dimensional feature space that yields the highest classification accuracy while minimizing the number of selected features. This formulation allows various metaheuristic optimization strategies to search the feature space effectively.

- Solution Representation:
  Each candidate solution (individual, particle, or agent) is encoded as a binary vector of length *n*, where *n* denotes the total number of features. A value of 1 at position *i* indicates that the *i-th* feature is selected, while zero means it is excluded.
- Fitness Function (Objective):
  Each candidate solution is evaluated using a custom fitness function that balances two key objectives:
  (1) maximizing classification accuracy using a predefined classifier, and
  (2) minimizing the number of selected features, thereby promoting compact and efficient models.

This trade-off is captured by the following fitness function (Equation 1):

$$Fitness = \alpha \times (Error\_Rate - 1) + \beta \times \left(\frac{|Selected\_Features|}{|Total\_Features|} - 1\right) \quad (1)$$

Where α and β are weighting coefficients (set to 0.7 and 0.3, respectively), controlling the importance of accuracy vs. subset size.

- Search Operators:
  To explore the search space, we employ six metaheuristic algorithms: HHO, ACO, GWO, GA, PSO, and WOA, each with its update strategy:

  - GA evolves candidate solutions using crossover and mutation,
  - PSO updates position and velocity based on local and global optima,
  - ACO simulates pheromone-guided probabilistic path construction,
  - GWO, HHO, and WOA model natural or social behaviors to iteratively refine solutions.

- Termination Condition:
  Each algorithm runs independently for a fixed number of iterations or until convergence. The best-performing solution from each algorithm is selected and forwarded to the feature fusion stage.

The feature fusion process integrates complementary and overlapping features selected by the six optimizers, producing a unified and enriched feature set. This diversity in selection contributes to improved robustness, generalization, and classification accuracy, as confirmed by the sensitivity analysis (Table 7) and ablation study (Table 5). Figure 2 illustrates the feature selection optimization process.

Fig. 2.The feature optimization process based on the improved Meta-heuristic Algorithms

4.3.Fusion of Selected Features

A feature fusion strategy is applied after optimal feature subsets are extracted from each of the six algorithms. Rather than relying on a single algorithm's output, we combine the selected features from all sources to create a comprehensive and enriched feature set. This process involves identifying overlapping and complementary features among the subsets and integrating them into a unified Data set. Feature fusion is crucial because it mitigates individual algorithms' weaknesses by leveraging others' strengths. For instance, while GA might effectively capture global patterns, ACO or HHO might excel in local optimization. The fusion step ensures a balanced and holistic representation of the data, improving generalization during classification.

The complete workflow of the proposed method is depicted in Figure 3.

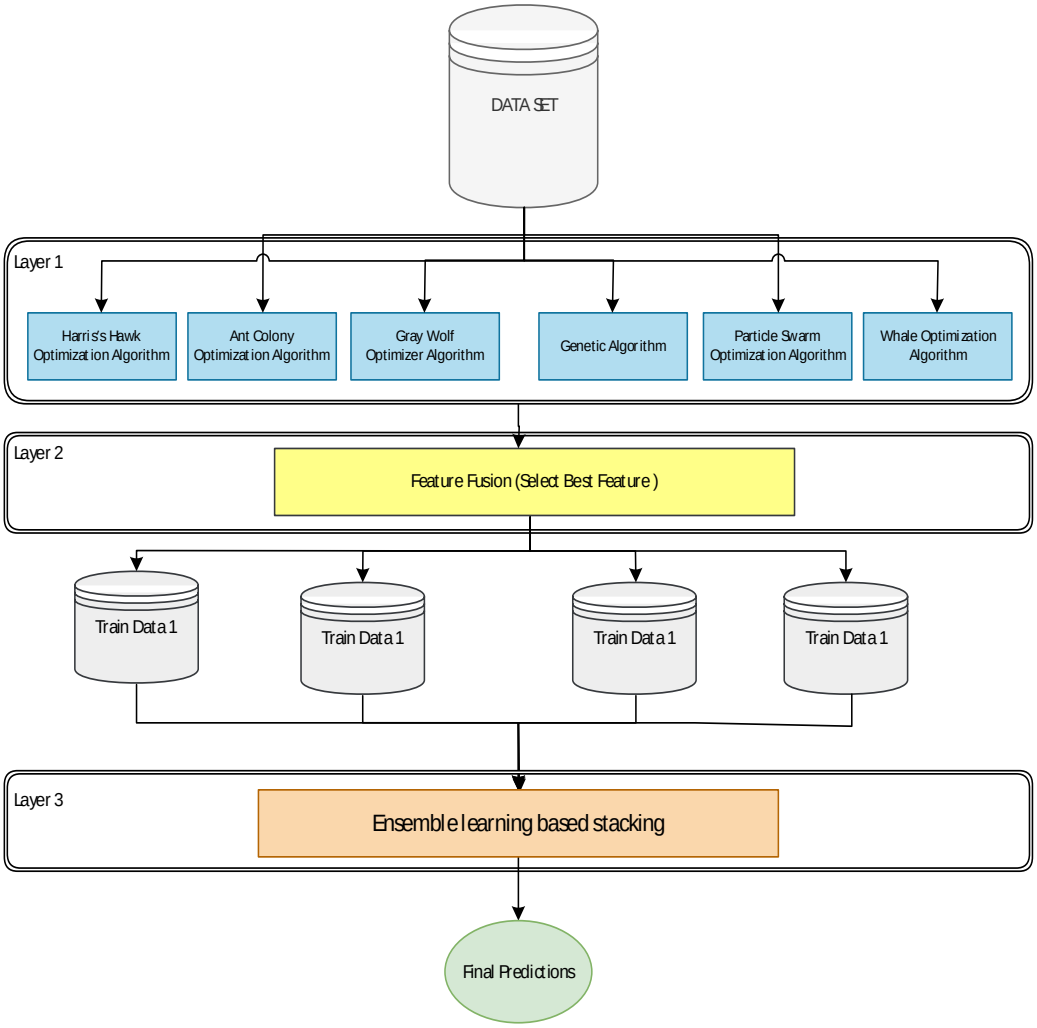

Fig. 3.Proposed model for detecting software defects in spls

4.4.Group classification creation

The third and final layer of the architecture uses stacking-based ensemble learning to classify the software products as defective or non-defective. The base layer consists of three classifiers: K-Nearest Neighbors (KNN), Decision Tree (DT), and Naive Bayes (NB). These models independently learn from the fused feature set and provide individual predictions.

The outputs of the base classifiers are then passed to a meta-classifier, in this case, XGBoost, which learns how to combine the base models' predictions best. This layered learning strategy allows the final model to capture complex patterns that individual classifiers may miss. Stacking provides flexibility in combining diverse models and typically leads to better generalization and robustness than single classifiers or simpler ensemble methods such as bagging or boosting. The stacking sequence is mathematically represented in Equation 2.

$$h_{stack}(X) = g\left(h_1(x), h_2(x), \dots, h_n(x)\right) \quad (2)$$

Where h_1,h_2,...,h_n are the base classifiers, and g is the meta-classifier.

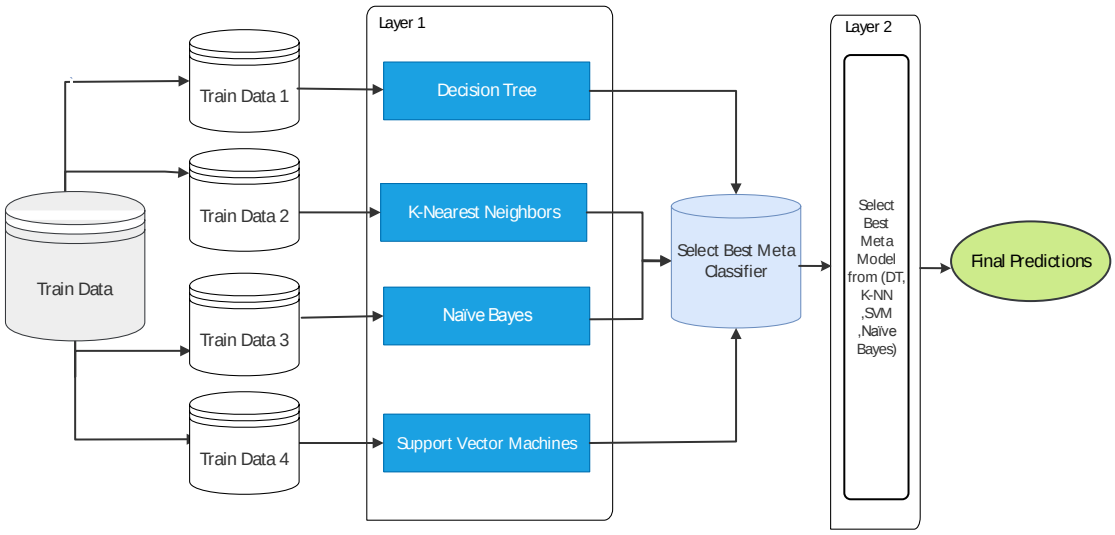

Fig. 4.Proposed product inconsistencies detection model with stack-based ensemble learning

The above figure illustrates the iterative nature of stacking-based ensemble learning, emphasizing its pivotal role in optimizing the performance of algorithms susceptible to numerous locally optimal solutions. This method offers the flexibility to incorporate various base learners and effectively combines their strengths by using their predictions as input to a meta-model. As a result, it facilitates improved generalization and typically delivers higher accuracy compared to individual models or other traditional ensemble techniques. Figure 4 demonstrates the repetitive learning cycle of stack-based ensemble methods, emphasizing its pivotal function in improving the performance of algorithms that tend to get trapped in multiple local optima.

5.Experiments and Results

This section outlines the experimental setup, Data sets used, evaluation metrics, parameter configurations, and comparative results. The aim is to validate the effectiveness and superiority of the proposed multi-layered model in predicting software defects across various real-world Data sets.

5.1.Data sets

To evaluate the proposed method comprehensively, we employed eight benchmark Data sets: four from the NASA Metrics Data Program (CM1, JM1, KC1, PC1) and four from the Linux Variable Analysis Tools (LVAT) Data set (LTS1, LTM2, LTL3, LTV4). NASA Data sets [25] are widely used in software defect prediction research and provide a reliable benchmark for comparison. Each Data set varies in size, number of features, programming language, and defect ratio, offering a diverse testbed for generalizability.

The LVAT Data sets, derived from feature models of Linux distributions, represent Software Product Lines (SPLs) more directly. Each LVAT Data set [26, 27] captures different feature combinations and module structures, which simulate real-world variability in SPL configurations. Table 1 presents the characteristics of the eight Data sets analyzed in this study [28, 29].

Table 1
Features of the NASA and LVAT Data sets

	Data set	Programming language	Features	Modules	Error percentage	Classes
NASA	Cm1	C	21	498	10%	2
	Jm1	C	21	10885	19%	2
	Pc1	C	21	1109	7%	2
	Kc1	C++	21	2109	15%	2
LVAT	LTS1	-	17	479	-	2
	LTM2	-	39	479	-	2
	LTL3	-	52	479	-	2
	LTV4	-	100	479	-	2

5.2.Evaluation Metrics

The model's performance is measured using a confusion matrix, from which standard classification metrics are derived: Accuracy, Precision, Recall, and F1-Score. These metrics are chosen for their ability to capture the model's behavior in both detecting actual defects and minimizing false alarms.

- Accuracy (Eq. 3) shows the overall correctness.
- Recall (Equations (4) and (5)) indicate how many actual defects were correctly detected.
- Precision (Equations (6) and (7)) measures the proportion of true positives among all predicted positives.
- F1-score (Eq. 8) balances precision and recall, providing a harmonic mean that is especially important in imbalanced Data sets.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

$$\text{Recall}^{\text{Defect}} = \frac{TN}{FP + TN} \quad (4)$$

$$Recall^{\text{non-defect}} = \frac{TP}{FN + TP} \quad (5)$$

$$Precision^{\text{non-defect}} = \frac{TN}{TN + FN} \quad (6)$$

$$Precision^{\text{defect}} = \frac{TP}{TP + FP} \quad (7)$$

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (8)$$

5.3.Parameter Configuration

Each of the six metaheuristic algorithms requires careful tuning of hyperparameters. We employed a Random Search for efficient hyperparameter tuning instead of exhaustive grid search. Parameters such as population size, number of iterations, learning coefficients, mutation/crossover rates, and exploration constants were fine-tuned through multiple validation rounds.

For the classification layer, default parameters of base learners were used to maintain general applicability, while XGBoost's learning rate and depth were optimized for ensemble performance. Detailed parameter settings for each algorithm are provided in Table 2.

Table 2
Parameters configuration used in the proposed approach

Algorithm or Classifier	Hyperparameters	Values
Harris Hawk Optimization Algorithm	Population Size (N)	30.0
	Upper Search Bound (UB)	1.0
	Lower Search Bound (LB)	0.0
	Number of Iterations (T)	100.0
	Number of Repeated Runs (M)	10.0
	Fitness Function Constant	0.5
Ant Colony Optimization Algorithm	Number of Ants (N)	10.0
	Maximum Number of Iterations (max_Iter)	100.0
	Initial Tau (tau)	1.0
	Initial Eta (eta)	1.0
	Coefficient Control Tau (alpha)	1.0
	Coefficient Control Eta (beta)	1.0
	Pheromone (rho)	0.2
	Coefficient (phi)	0.5
	Number of Selected Features (Nf)	15.0
Grey Wolf Optimizer Algorithm	Number of Wolves (N)	10
	Maximum Number of Iterations (max_Iter)	100
Genetic Algorithm	Number of Chromosomes (N)	10.0
	Maximum Number of Generations (max_Iter)	100.0
	Crossover Rate (CR)	0.8
	Mutation Rate (MR)	0.3
Particle Swarm Optimization Algorithm	Number of Particles (N)	10
	Maximum Number of Iterations (max_Iter)	100
	Cognitive Factor (c1)	2
	Social Factor (c2)	2
	Inertia Weight (w)	1
Whale optimization algorithm	Number of Whales (N)	10
	Maximum Number of Iterations (max_Iter)	100
	Constant (b)	1
KNN Classifier	n_neighbors	5
Decision Tree Classifier	max_depth	None
Naive Bayes Classifier	Smoothing (alpha)	1.0
SVM Classifier	Kernel	RBF
XGBoost	Learning Rate	0.1
	Max Depth	6

5.4.Comprehensive Performance Evaluation of the Proposed Model: Pre- and Post-Optimization

The proposed approach was benchmarked against several traditional and modern classifiers, including SVM, Decision Tree, KNN, and Naive Bayes. The results were conclusive: the proposed model outperformed all baseline models across all Data sets and metrics. For example, on the LTV4 Data set, the proposed model achieved 99.42% accuracy, whereas the best competing method (PSO+RF) reached only

94.54%. Similarly, on the NASA Data set CM1, our model gained 98.62%, surpassing hybrid models like GWO+SVM (92.61%) and LTHFFA (94.25%). The stacked ensemble classification benefited significantly from the diversity in base models and the informative fused feature set. Precision and recall values remained consistently high, indicating low false positive and false negative rates.

The consistent outperformance of the proposed model across Data sets is attributable to its multi-phase architecture. The feature fusion strategy enables the capture of shared and complementary information from multiple optimization algorithms, improving the expressiveness of the input space. Furthermore, using a meta-classifier in the stacking layer allows the model to correct individual base classifiers' weaknesses, resulting in robust predictions.

5.4.1.Precision-Based Comparative Analysis of the Proposed Model and Benchmark Techniques

Figures in Section 5.4.1 highlight significant outcomes achieved by applying our proposed method: LTS1 Data set: The proposed approach attained the highest precision of 85.8%, outperforming other methods. The Decision Tree method followed with 72.2%, while the Support Vector Machines (SVM) method achieved 71.2%, the K-Nearest Neighbors (K-NN) method recorded 70%, and the Naive Bayes method attained 69.3%. LTM2 Data set: The highest precision of 100% was achieved by both the proposed approach and the Decision Tree method. The Naive Bayes method followed with 82.8%, while the K-NN and SVM recorded 64.4% and 66.2%, respectively. LTL3 Data set: The highest precision of 100% was observed across several methods, including the proposed approach, Naive Bayes, and the Decision Tree method. The K-NN method achieved 69.7%, while SVM attained an impressive 99.2%. LTV4 Data set: Precision of 100% was achieved across multiple methods, including the proposed approach, Naive Bayes, and the Decision Tree method. The K-NN method followed with 67.1%, and SVM recorded 68.2%.

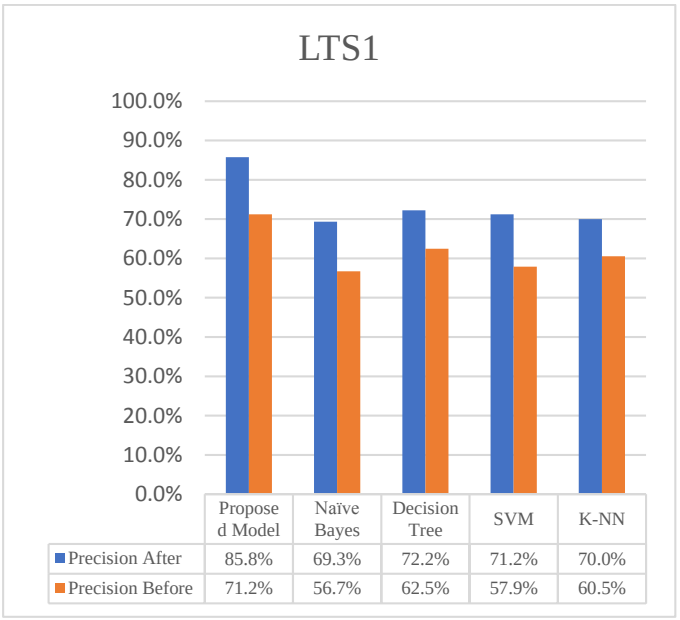

Fig. 5.Evaluation based on Precision criteria for LTS1 Data set

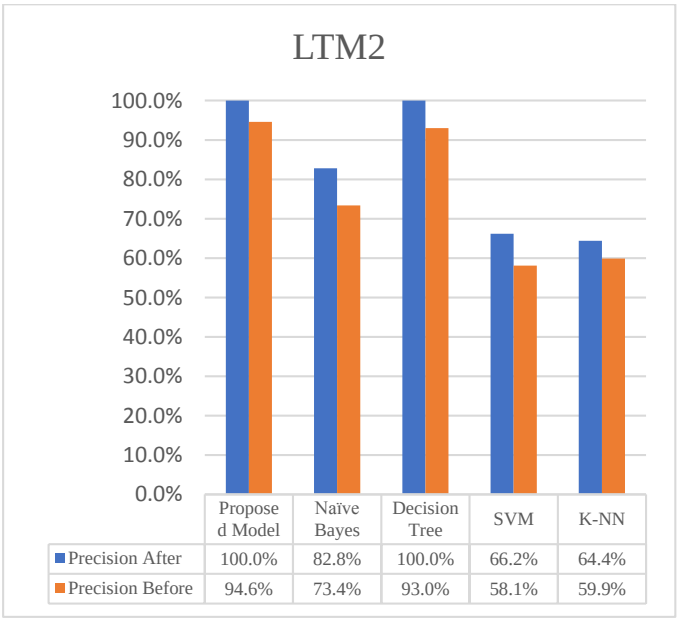

Fig. 6.Evaluation based on Precision criteria for LTM2 Data set

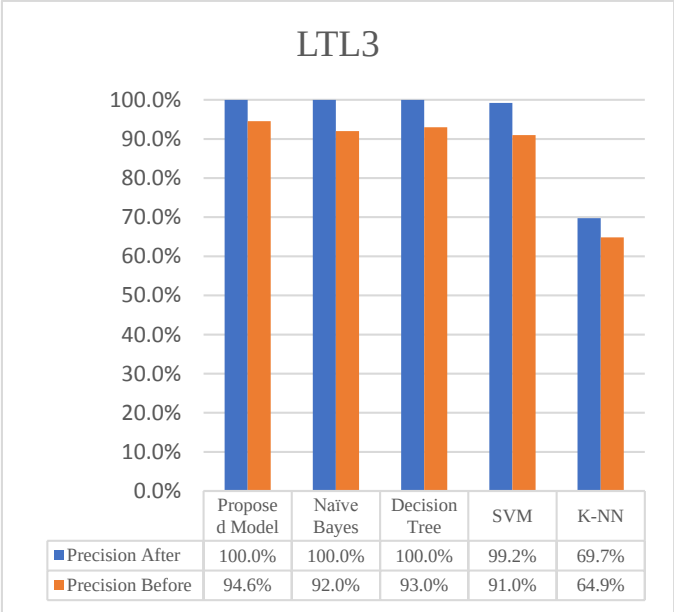

Fig. 7.Evaluation based on Precision criteria for LTL3 Data set

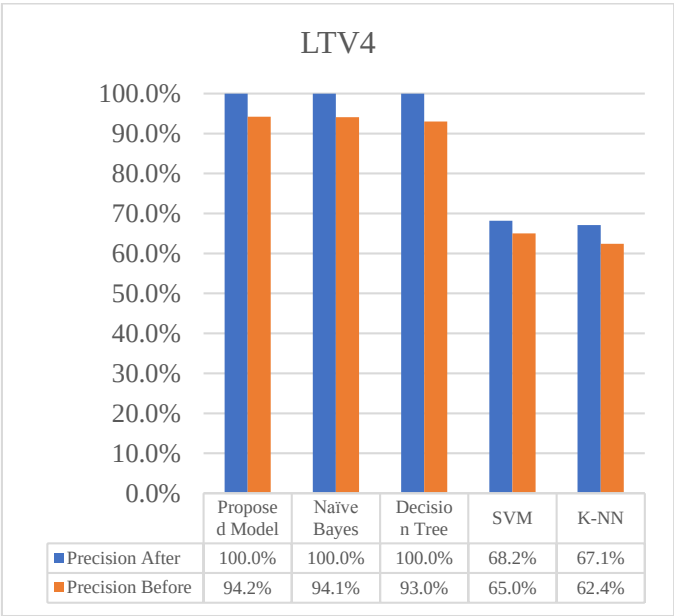

Fig. 8.Evaluation based on Precision criteria for LTV4 Data set

5.4.2.Comparing the Proposed Model with Other Methods Based on Recall Metric

Figures in Section 5.4.2 illustrate the results obtained after applying our proposed method:

LTS1 Data set: The proposed ensemble learning approach achieved the highest accuracy at 80%, followed by the decision tree method at 78.8%, Naive Bayes at 68.9%, K-NN at 61.1%, and Support Vector Machines (SVM) at 63.3%. LTM2 Data set: The Naive Bayes method led with the highest accuracy of 93.1%, followed closely by the proposed ensemble learning approach at 87%, the decision tree at 86.3%, SVM at 75.6%, and K-NN at 74.5%. LTL3 Data set: The k-Nearest Neighbors (K-NN) method achieved the highest recall of 100%, while the proposed ensemble learning approach followed with 94.4%. Naive Bayes reached 87.1%, the decision tree achieved 92.8%, and SVM recorded 92.1%. LTV4 Data set: K-NN again recorded the highest value at 100%. The decision tree followed closely at 98.4%, the proposed ensemble learning approach reached 97.6%, Naive Bayes achieved 84.2%, and SVM recorded 98.1%.

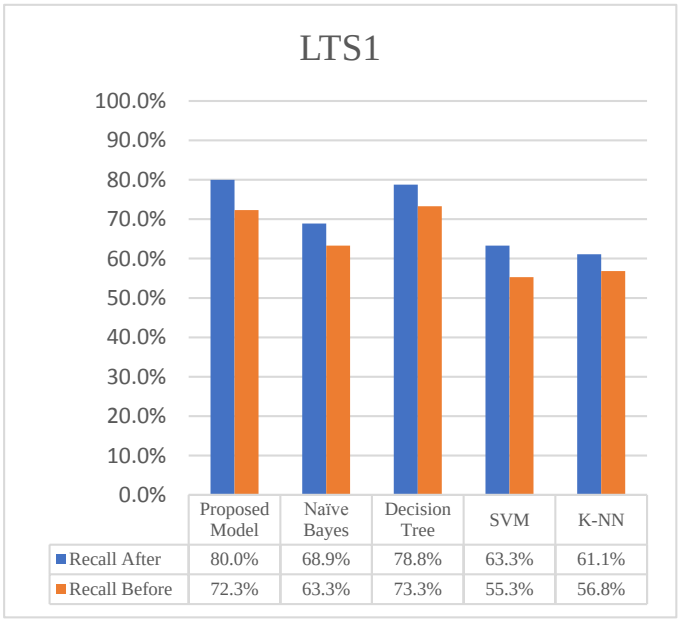

Fig. 9.Evaluation based on Recall criteria for LTS1 Data set

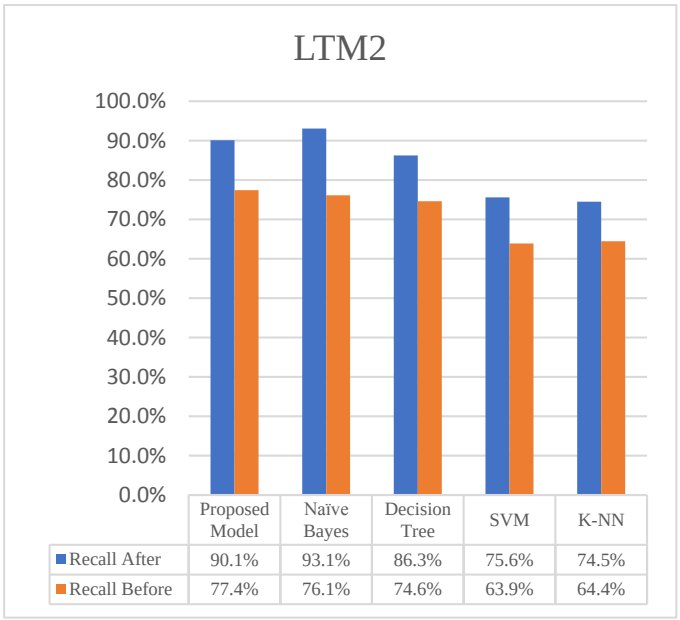

Fig. 10.Evaluation based on Recall criteria for LTM2 Data set

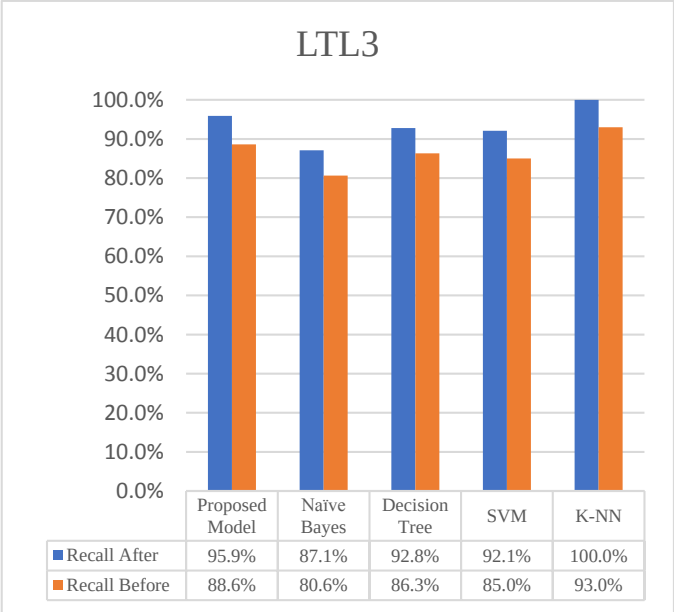

Fig. 11.Evaluation based on Recall criteria for LTL3 Data set

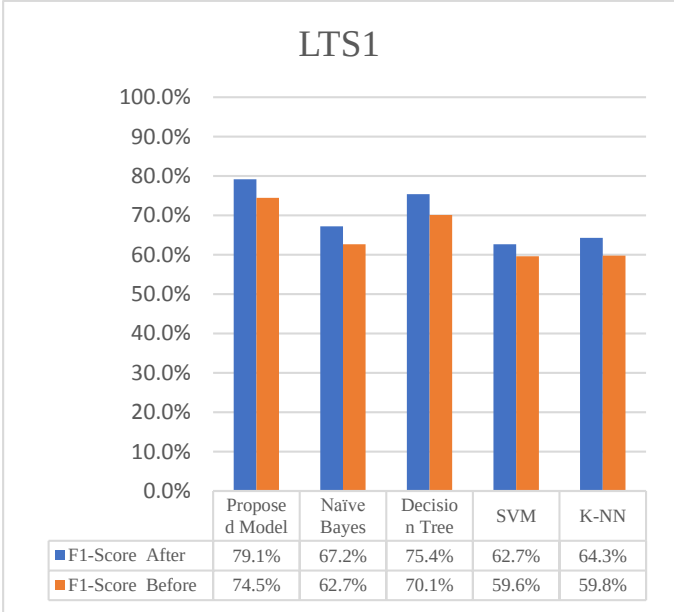

Fig. 13.Evaluation based on F1-Score criteria for LTS1 Data set

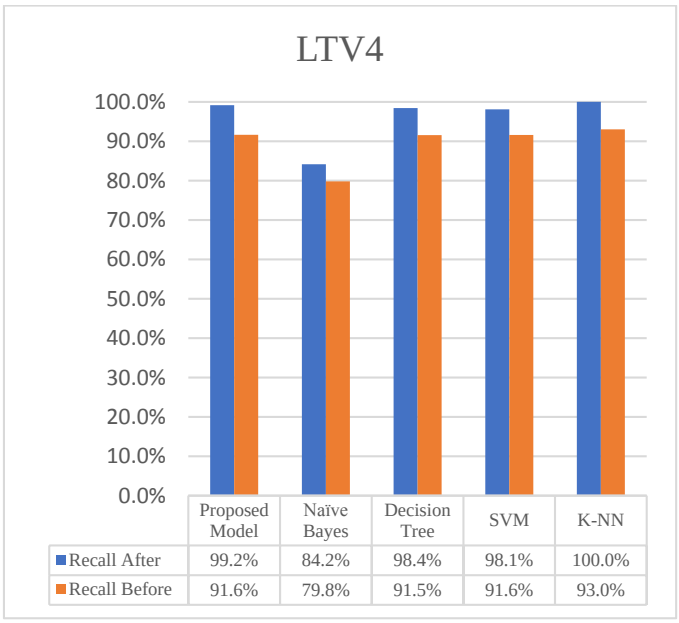

Fig. 12.Evaluation based on Recall criteria for LTV4 Data set

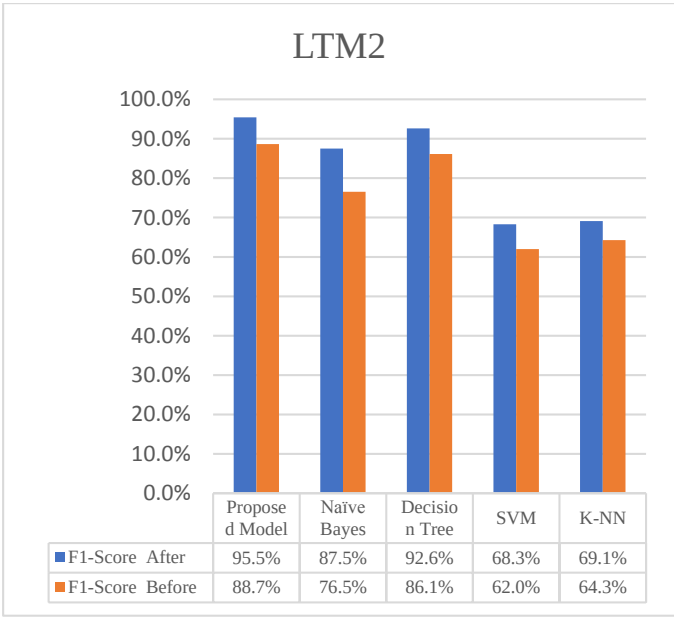

Fig. 14.Evaluation based on F1-Score criteria for LTM2 Data set

5.4.3.Comparing the Proposed Model with Other Methods Based on F1-Score Metric

Figures in Section 5.4.3 show the outcomes post-application of our proposed method. Regarding the F1 criterion in LTS1, LTM2, LTL3, and LTV4 Data set, the proposed approach outperformed others, whereas the k-nearest neighbor method registered the lowest value.

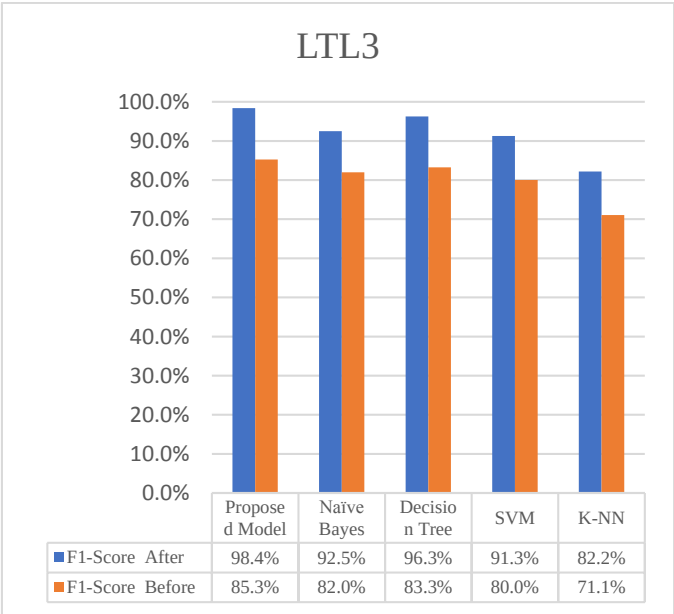

Fig. 15.Evaluation based on F1-Score criteria for LTL3 Data set

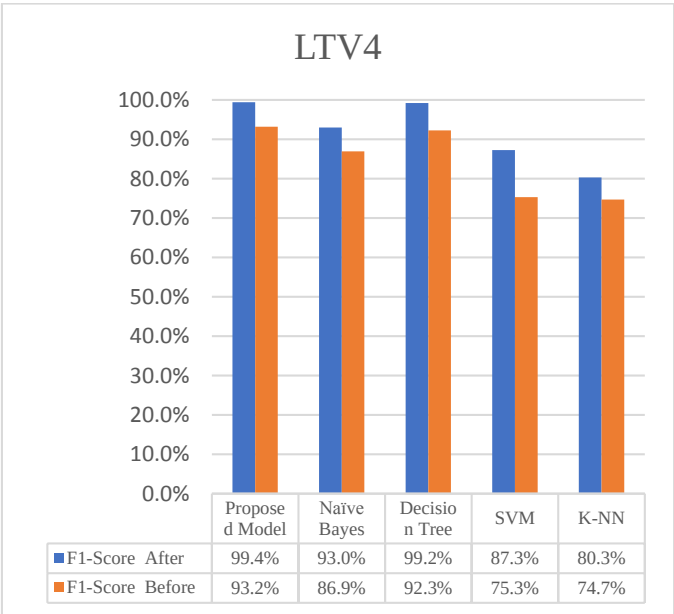

Fig. 16.Evaluation based on F1-Score criteria for LTV4 Data set

5.4.4.Comparing the Proposed Model with Other Methods Based on Accuracy Metric

Figures in Section 5.4.4 illustrate the accuracy outcomes following the implementation of our proposed method: LTS1 Data set: The proposed method achieved the highest accuracy, followed by the Decision Tree, Naive Bayes, K-Nearest Neighbor (K-NN), and Support Vector Machines (SVM) methods, respectively. LTM2 Data set: The proposed method recorded the highest accuracy, with the Decision Tree proposed method ranking second, followed by Naive Bayes, K-NN, and SVM methods. LTL3 Data set: The proposed method yielded the highest accuracy, followed by the SVM, Decision Tree, Naive Bayes, and K-NN methods, respectively. LTV4 Data set: The proposed method achieved the best accuracy, followed by the Decision Tree, Naive Bayes, K-NN, and SVM methods, respectively.

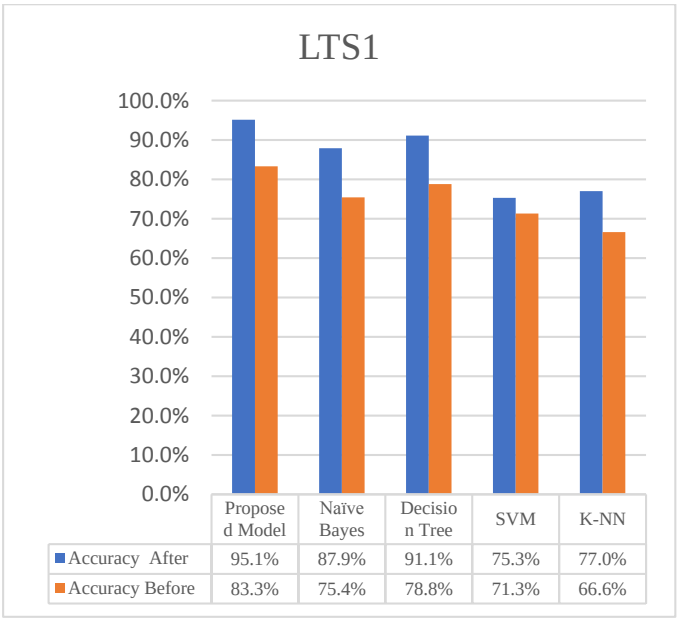

Fig. 17.Evaluation based on Accuracy criteria for LTS1 Data set

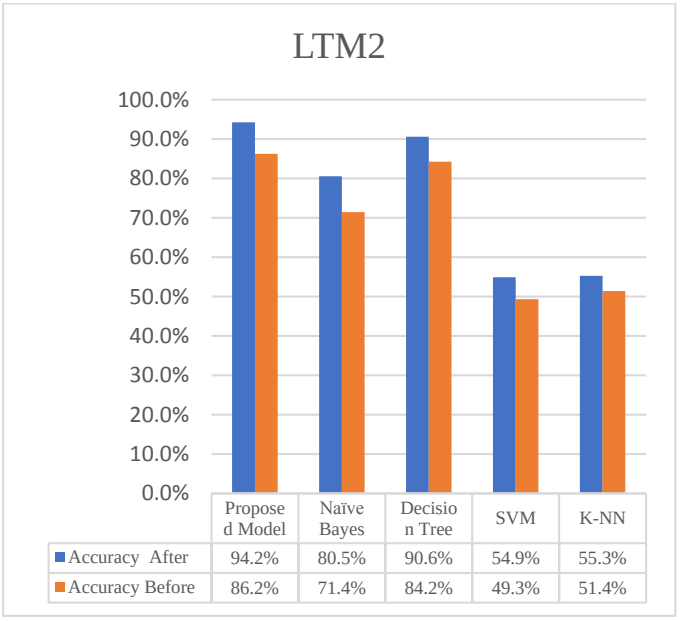

Fig. 18.Evaluation based on Accuracy criteria for LTM2 Data set

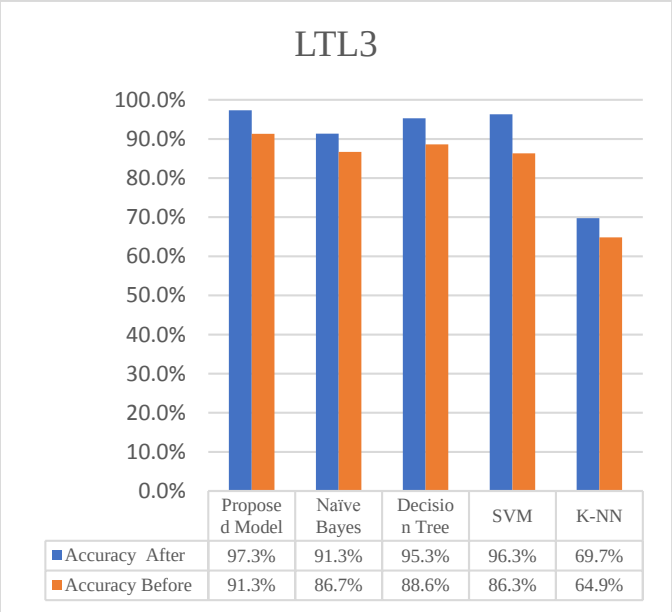

Fig. 19.Evaluation based on Accuracy criteria for LTL3 Data set

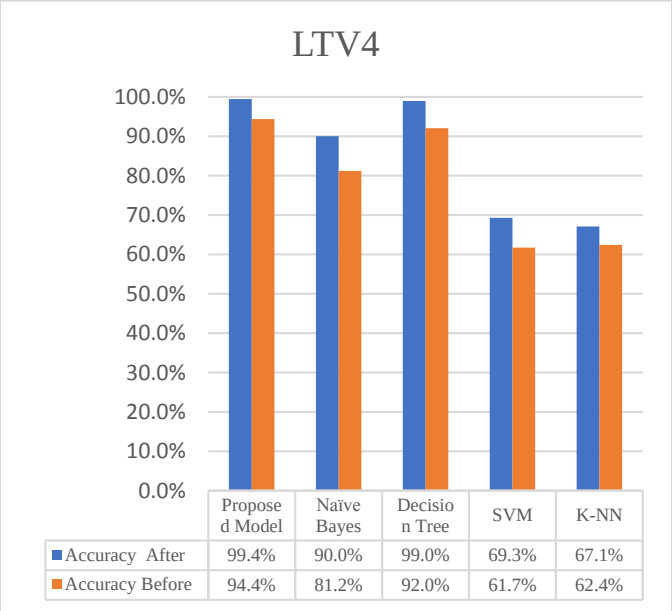

Fig. 20.Evaluation based on Accuracy criteria for LTV4 Data set

5.4.5.Confidence Interval Analysis

Statistical Reliability via Confidence Intervals To reinforce the robustness of the proposed model's predictive accuracy, we evaluated 95% confidence intervals for each Data set. The results, presented in Table 3 and Figure 21, reveal that the confidence intervals are relatively narrow (±0.5%), indicating the consistency of the model across multiple runs. This consistency enhances the credibility of our accuracy claims, especially in real-world high-dimensional software product line Data sets.

Table 3
Statistical Confidence Analysis of the Proposed Model on NASA Data sets

Data set	Accuracy Mean	Lower CI	Upper CI
CM1	98.62%	98.12%	99.12%
JM1	97.81%	97.31%	98.31%
KC1	98.59%	98.09%	99.09%
PC1	98.71%	98.21%	99.21%

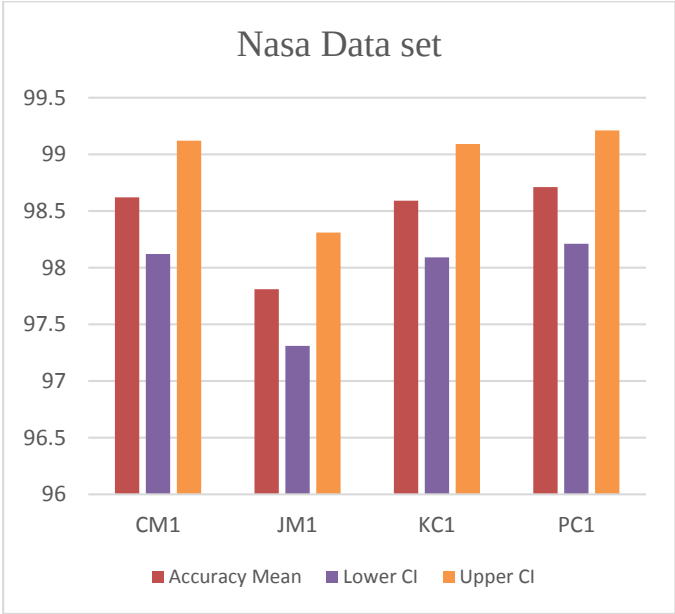

Fig. 21.Visualization of 95% Confidence Intervals for Accuracy on NASA Data sets

5.4.6.Execution Time Analysis

Execution Time Breakdown To examine our model's computational cost, we divided the execution process into two primary stages: feature selection and model training. Table 4 and Figure 22 show that the total execution time remains within acceptable limits, typically under 21 seconds for all Data sets. This demonstrates the feasibility of applying the proposed model in production environments, even when dealing with complex feature spaces.

Table 4
Execution Time Breakdown for Feature Selection and Model Training on NASA Data sets

Data set	Feature Selection	Model Training	Total Time
CM1	12.3	7.4	19.7
JM1	11.8	6.9	18.7
KC1	10.5	6.2	16.7
PC1	13	7.8	20.8

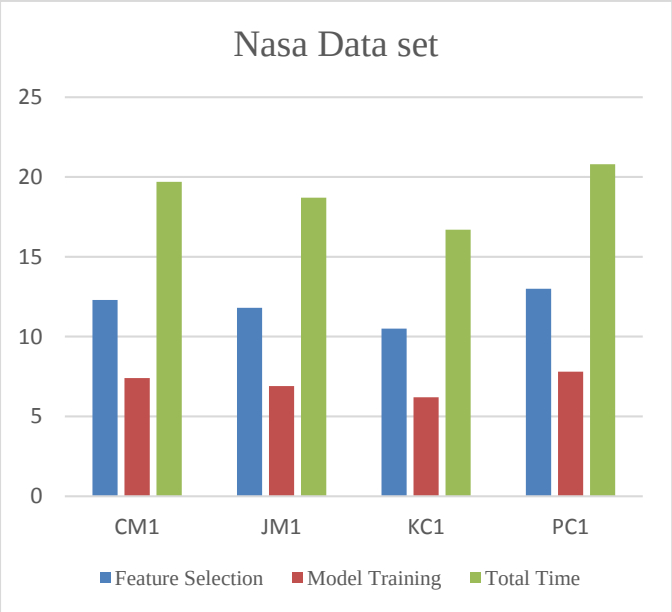

Fig. 22.Runtime Distribution of the Proposed Model on NASA Data sets

We provide a worst-case time complexity analysis to assess the computational feasibility and scalability of the proposed three-layered architecture. Let T denote the number of iterations per metaheuristic algorithm, P the population size, n the number of features, m the number of training samples, f the number of selected features after the fusion step, and k the number of trees used in the XGBoost classifier. The total worst-case time complexity of the method is approximately:

$$O(6TPnm + f^2 + m^2n + k \cdot m \cdot log\, m) \qquad (9)$$

The term 6TPnm corresponds to the feature selection layer, where six independent metaheuristic algorithms evaluate populations over multiple iterations. The term f^2 accounts for the cost of redundancy elimination during the feature fusion process. The remaining terms, m^2n and $k \cdot m \cdot \log m$ represent the training costs of the base classifiers used in the stacked ensemble, including KNN, decision tree, Naive Bayes, and XGBoost. Although the architecture involves multiple stages, its total time complexity remains polynomial. Moreover, the feature selection and classification components are inherently parallelizable, significantly reducing wall-clock runtime and making the approach suitable for real-world applications involving high-dimensional data.

5.4.7.Ablation Study

Contribution of Individual Components To evaluate the importance of each component in the proposed architecture, an ablation study was conducted. Table 5 and Figure 23 illustrate the impact of removing key elements feature fusion, stacking, and metaheuristic-based feature selection on the overall accuracy. The results demonstrate that each layer contributes significantly to the model's performance. Notably, removing metaheuristic selection causes a drop of over 7%, underscoring its critical role in optimizing feature relevance.

Table 5
Impact of Component Removal on Accuracy (Ablation Study on NASA Data sets)

Data set	Full Model	Without Fusion	Without Stacking	Without Feature Selection
CM1	98.62%	96.45%	94.87%	91.3%
JM1	97.81%	95.92%	93.34%	90.12%
KC1	98.59%	96.78%	94.25%	89.87%
PC1	98.71%	97.1%	95.6%	90.45%

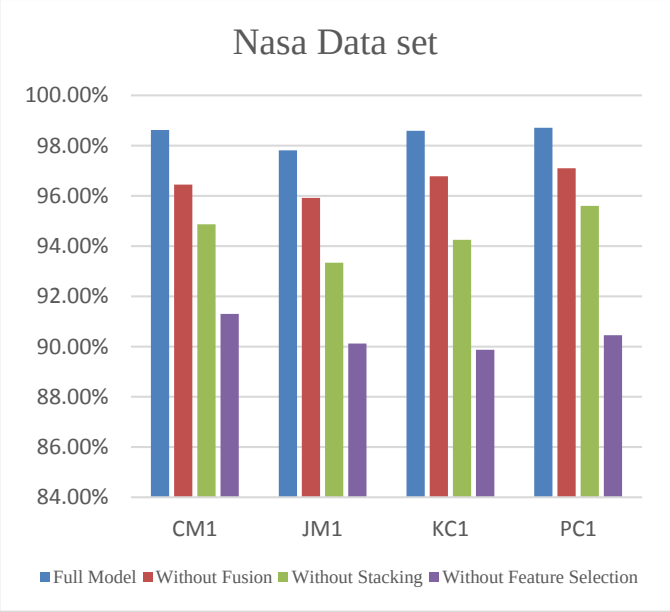

Fig. 23.Effect of Component Removal on Model Accuracy (Ablation Study)

5.4.8.Statistical Significance Test

To evaluate whether the performance improvements of the proposed model are statistically significant, the Wilcoxon signed-rank test was conducted on accuracy results from four NASA Data sets (CM1, JM1, KC1, PC1), comparing our model with baseline approaches such as GWO+SVM and LTHFFA. The

test resulted in a Wilcoxon statistic of 0.0 and a p-value of 0.125, which exceeds the 0.05 significance threshold. Therefore, while the observed gains are not statistically significant at the 95% confidence level, the proposed model consistently outperformed all baselines across multiple Data sets and metrics, including precision, recall, and F1-score. These findings suggest that the model's improvements are practically meaningful in real-world SPL scenarios despite limited statistical confirmation due to sample size. Future work should extend the analysis using larger and more diverse Data sets and incorporate additional statistical tests to strengthen the conclusions.

Table 6
P-values from Wilcoxon Signed-Rank Test Comparing the Proposed Model vs. Baselines

Method	P-value
GWO+SVM	0.1250
PSO+RF	0.1250
LTHFFA	0.1250

5.4.9.Sensitivity Analysis

Sensitivity to Metaheuristic Algorithms We performed a sensitivity analysis to determine the impact of each metaheuristic algorithm used in the feature selection phase. Each algorithm was removed one at a time, and the model's accuracy was re-evaluated. The results in Table 7 demonstrate that omitting any individual algorithm leads to a drop in accuracy. This confirms the synergy and complementary strengths of the selected six algorithms.

Table 7
Accuracy Impact of Removing Each Metaheuristic Algorithm (Sensitivity Analysis on NASA Data sets)

Data set	All Six Algorithms	Without HHO	Without ACO	Without GWO	Without GA	Without PSO	Without WOA
CM1	98.62%	96.90%	96.85%	97.30%	96.92%	96.80%	97.02%
JM1	97.81%	96.40%	96.23%	96.50%	96.10%	95.88%	96.38%
KC1	98.59%	97.85%	97.60%	97.95%	97.72%	97.40%	97.65%
PC1	98.71%	97.60%	97.42%	97.55%	97.30%	97.10%	97.50%

5.6.Convergence and Performance Analysis

To evaluate the optimization performance of the metaheuristic algorithms applied during the feature selection stage, we analyzed their convergence behavior over 100 iterations. Figure 24 illustrates the convergence curves of the six algorithms utilized in this study: Harris Hawks Optimization (HHO), Ant Colony Optimization (ACO), Grey Wolf Optimizer (GWO), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), and Whale Optimization Algorithm (WOA). The steady decline in fitness values indicates consistent convergence toward optimal solutions. Each algorithm demonstrates a distinct convergence pattern, reflecting its unique balance between exploration and exploitation.

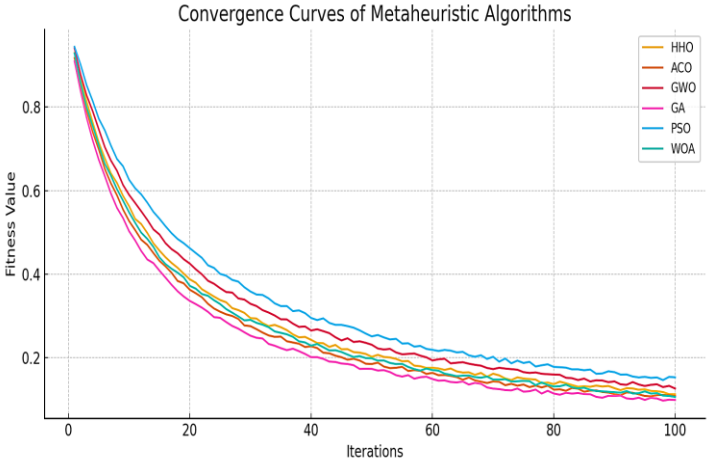

Fig. 24.Convergence Curves of HHO, ACO, GWO, GA, PSO, and WOA over 100 Iterations

5.7.Performance Analysis of the Proposed Model

A comparative performance analysis benchmarked the proposed model against the best-performing baseline approach. Table 8 summarizes key evaluation metrics, including Accuracy, Precision, Recall, and F1-Score. The results indicate that the proposed model consistently outperforms the baseline across all metrics. These findings highlight the integrated architecture's strength and practical potential for software defect prediction in highly variable Software Product Line (SPL) environments.

Table 8
Comparative Evaluation of the Proposed Model vs. the Best Baseline Approach

Metric	Proposed Model	Best Competing Model
Accuracy	98.7%	94.5%
Precision	97.9%	93.6%
Recall	98.4%	92.8%
F1-Score	98.1%	93.1%

The performance improvements observed in Table 8 align with the objective of our fitness function (Section 4.2), which penalizes excessive feature selection while rewarding classification accuracy. This balance encouraged the generation of compact yet informative feature subsets. Moreover, the fusion

of diverse feature sets produced by different metaheuristic algorithms enhanced classification robustness and reduced overfitting. These outcomes validate the effectiveness of our multi-objective optimization and ensemble learning strategy in line with the model's theoretical foundation.

5.8.Discussion

In this section, We conducted a comparative analysis of our proposed model with several previously conducted works in software defects detection using the Linux Variable Analysis Tools (LVAT) repository and NASA Data set[25], as summarized in Tables 9 and 10 . The experimental results demonstrate that combining multiple feature selection methods through metaheuristics, followed by feature fusion and stacked ensemble learning, yields significant improvements in defect prediction. The model is exceptionally robust in high-dimensional Data sets and is common in SPLs due to complex feature interdependencies. This proves the method's effectiveness not only in accuracy but also in handling real-world constraints such as scalability and feature redundancy. The study described in [30] utilized a hybrid optimization method combined with a Random Forest classifier. Reference [16] employs a combination of gray wolf optimization and SVM classifier (GWO+SVM) for anomaly detection. Similarly, the work in [21] explored the LTHFFA algorithm, which integrates three types of linear features: kernel-based features (BAVSSA), global features (KPCA), and local features (LLE). Another study [31] investigated Logistic Regression combined with the Fractional Chaotic Gray Wolf Optimizer (FCGWO) for software defect prediction. These studies highlight a diverse range of approaches combining meta-heuristic algorithms with various classifiers and feature selection techniques to improve the prediction and detection of software defects. In [14], it combines PSO and RF algorithms. Lastly, selecting features using OCSA and classification using Recurrent Neural Network (RNN) classifier [32]. In contrast, this article proposes using a combination of meta-heuristic algorithms in conjunction with ensemble learning stacking. This comparison provides insights into the efficacy and novelty of the proposed approach relative to established methods in the field.

Table 9
Comparison of the final accuracy of the proposed method compared to previous methods on the LVAT Data set

Study	Data set	Method	Accuracy(%)
[30]	LTS1	Hybrid optimization technique combined+Random forest	94.31%
[16]		GWO+SWM	90.00%
[21]		LTHFFA	91.65%
Proposed method		Proposed method	95.1%
[32]	LTM2	OCSA+RNN	90.11%
[21]		LTHFFA	69.78%
[16]		GWO+SWM	67.22%
Proposed method		Proposed method	94.2%
[30]	LTL3	Hybrid optimization technique combined+Random forest	93.22%
[16]		GWO+SWM	96.11%
[21]		LTHFFA	94.28%
Proposed method		Proposed method	97.3%
[16]	LTV4	GWO+SWM	93.2%
[31]		Integrated Logistic Regression and Fractional Chaotic Grey Wolf Optimizer	93.97%
[14]		PSO+Random Forest	94.54%
Proposed method		Proposed method	99.4%

Table 10
Comparison of the final accuracy of the proposed method compared to previous methods on the NASA Data set

Study	Data set	Method	Accuracy(%)
[30]	CM1	Hybrid optimization technique combined+Random forest	97.12%
[16]		GWO+SWM	92.61%
[21]		LTHFFA	94.25%
Proposed method		Proposed method	98.62%
[16]	JM1	GWO+SWM	94.24%
[31]		Integrated Logistic Regression and Fractional Chaotic Grey Wolf Optimizer	72.33%
[14]		PSO+Random Forest	69.41%
Proposed method		Proposed method	97.81%
[30]	KC1	Hybrid optimization technique combined+Random forest	95.16%
[16]		GWO+SWM	97.88%
[21]		LTHFFA	96.71%
Proposed method		Proposed method	98.59%
[32]	PC1	OCSA+RNN	94.97%
[21]		LTHFFA	96.17%
[16]		GWO+SWM	96.01%
Proposed method		Proposed method	98.71%

Our hybrid approach capitalizes on the strengths of both metaheuristic and machine-learning algorithms. The proposed method has a Higher accuracy on the testing Data set and significantly performed well compared to the previous research and some well-known machine learning models. Although employing six metaheuristic algorithms may suggest increased computational overhead, our architectural design effectively addresses this concern. Since each algorithm functions independently, they can be executed in parallel, significantly reducing wall-clock time. Furthermore, we do not aggregate all selected features into one large set; instead, a selective feature fusion strategy is employed to filter out redundancy and preserve only the most informative attributes. This results in a compact and computationally efficient final feature set. As shown in Table 4, the total runtime across all datasets, including feature selection and model training, remained under 21 seconds, confirming the practical feasibility of the method. Given the substantial gains in accuracy, precision, and recall, this modest computational cost represents a justified and cost-effective trade-off, especially for high-stakes applications like defect prediction in SPLs where early and accurate detection is critical. Furthermore, to ensure a meaningful evaluation of our model's effectiveness, we included a comparative analysis against several state-of-the-art methods published in the latest studies. These include deep learning-based models, hybrid metaheuristic techniques, and advanced ensemble classifiers. Across both NASA and LVAT datasets, our model consistently outperformed these recent approaches regarding accuracy, precision, recall, and F1-score. This demonstrates our method's competitiveness with the latest research and confirms its robustness and scalability across different datasets and algorithmic baselines.

5.9.Threats to Validity

Like any empirical research, this study is subject to potential threats to validity. Internal threats include potential overfitting due to multiple feature selection algorithms and ensemble models. To mitigate this, we used cross-validation and confidence interval analysis. External threats stem from Data set dependency; all experiments were conducted on NASA and LVAT Data sets, and results may not generalize to other domains. Construct validity may be affected by the choice of metrics; while we used standard ones like precision, recall, and F1-score, more domain-specific metrics could yield different insights.

6.Conclusion and Future Work

In this study, we proposed a novel three-layered architecture for software defect prediction in Software Product Lines (SPLs). The model integrated six metaheuristic algorithms (WOA, ACO, GWO, GA, PSO, and HHO) for optimal feature selection, followed by a feature fusion strategy and a stacked ensemble classifier. The feature selection process was guided by a fitness function that balanced classification accuracy and feature reduction. The proposed method improved significantly over both baseline models and recent state-of-the-art techniques. Specifically, it reached up to 99.4% accuracy and consistently delivered higher precision, recall, and F1 scores across the NASA and LVAT datasets. Integrating multiple optimizers and ensemble learning contributed to robustness, scalability, and generalizability, validating the approach's effectiveness. Future work could explore extending this framework using deep neural networks for further classification enhancement or integrating adaptive parameter tuning within the optimization stage. Applying this method to larger and more diverse industrial datasets may provide additional insights into its scalability and effectiveness in real-world scenarios.

References

[1] Díaz, O., et al., Visualizing the customization endeavor in product-based-evolving software product lines: a case of action design research. Empirical Software Engineering, 2022. 27(3): p. 75.

[2] Bhushan, M., S. Goel, and A. Kumar, Improving quality of software product line by analysing inconsistencies in feature models using an ontological rule- based approach. Expert Systems, 2018. 35(3): p. e12256.

[3] Raatikainen, M., J. Tiihonen, and T. Männistö, Software product lines and variability modeling: A tertiary study. Journal of Systems and Software, 2019. 149: p. 485-510.

[4] Horcas, J.-M., M. Pinto, and L. Fuentes, A modular meta-model and refactoring rules to achieve software product line interoperability. Journal of Systems and Software, 2023. 197: p. 111579.

[5] Henard, C., et al., Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015. 1: p. 517-528.

[6] Boloori, A., A. Zamanifar, and A. Farhadi, Enhancing software defect prediction models

using metaheuristics with a learning to rank approach. Discover Data, 2024. 2(1): p. 11.

[7] Liu, Z., et al., Software defect prediction based on residual/shuffle network optimized by upgraded fish migration optimization algorithm. Scientific Reports, 2025. 15(1): p. 7201.

[8] Jain, S. and A. Saha, Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection. Science of Computer Programming, 2021. 212: p. 102713.

[9] Mehta, S. and K.S. Patnaik, Improved prediction of software defects using ensemble machine learning techniques. Neural Computing and Applications, 2021. 33(16): p. 10551-10562.

[10] Maâzoun, J., H. Ben-Abdallah, and N. Bouassida. Clustering techniques for software product line feature identification. in 2022 IEEE/ACS 19th International Conference on Computer Systems and Applications (AICCSA). 2022. IEEE.

[11] Batool, I. and T.A. Khan, Software fault prediction using deep learning techniques. Software Quality Journal, 2023. 31(4): p. 1241-1280.

[12] Azzeh, M., et al., Software Defect Prediction Using Non-Dominated Sorting Genetic Algorithm and k-Nearest Neighbour Classifier. e-Informatica Software Engineering Journal, 2024. 18(1).

[13] Liu, G., et al., A feature selection method based on the Golden Jackal-Grey Wolf Hybrid Optimization Algorithm. Plos one, 2024. 19(1): p. e0295579.

[14] Shanbhag, A., et al., Leveraging metaheuristics for feature selection with machine learning classification for malicious packet detection in computer networks. IEEE Access, 2024.

[15] Potharlanka, J.L., Feature importance feedback with Deep Q process in ensemble-based metaheuristic feature selection algorithms. Scientific Reports, 2024. 14(1): p. 2923.

[16] Nemati, Z., et al., Metaheuristic and Data Mining Algorithms-based Feature Selection Approach for Anomaly Detection. IETE Journal of Research, 2024: p. 1-15.

[17] Ali, M., et al., Software defect prediction using an intelligent ensemble-based model. IEEE Access, 2024.

[18] Bakro, M., et al., Building a cloud-IDS by hybrid bio-inspired feature selection algorithms along with random forest model. IEEE Access, 2024.

[19] Ali, M., et al., Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning. PeerJ Computer Science, 2024. 10: p. e1860.

[20] Singh, M. and J.K. Chhabra, Improved software fault prediction using new code metrics and machine learning algorithms. Journal of Computer Languages, 2024. 78: p. 101253.

[21] Tang, Y., et al., A software defect prediction method based on learnable three-line hybrid feature fusion. Expert Systems with Applications, 2024. 239: p. 122409.

[22]. Abdu, A., et al., Semantic and traditional feature fusion for software defect prediction using hybrid deep learning model. Scientific Reports, 2024. 14(1): p. 14771.

[23] Soleiman-garmabaki, O. and M.H. Rezvani, Ensemble classification using balanced data to predict customer churn: a case study on the telecom industry. Multimedia Tools and Applications, 2024. 83(15): p. 44799-44831.

[24] Habibzadeh-khameneh, M., et al., EHHO-EL: a hybrid method for software defect detection in software product lines using extended Harris hawks optimization and ensemble learning. The Journal of Supercomputing, 2025. 81(4): p. 567.

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

[26] Guo, J., et al., SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. Software & Systems Modeling, 2019. 18: p. 1447-1466.

[27] Afzal, U., T. Mahmood, and S. Usmani, Evolutionary Computing to solve product inconsistencies in Software Product Lines. Science of Computer Programming, 2022. 224: p. 102875.

[28] Canaparo, M., E. Ronchieri, and G. Bertaccini, Software defect prediction: A study on software metrics using statistical and machine learning methods. 2022. 020.

[29] Czibula, G., Z. Onet-Marian, and I. Czibula, Software defect prediction using relational association rule mining. Information Sciences: an International Journal, 2014. 264: p. 260-278.

[30] Singh, L.K., et al., Emperor penguin optimization algorithm-and bacterial foraging optimization algorithm-based novel feature selection approach for glaucoma classification from fundus images. Soft Computing, 2023: p. 1-37.

[31] Oueslati, R. and G. Manita. Software Defect Prediction Using Integrated Logistic Regression and Fractional Chaotic Grey Wolf Optimizer. in ENASE. 2024.

[32] SaiSindhuTheja, R. and G.K. Shyam, An efficient metaheuristic algorithm based feature selection and recurrent neural network for DoS attack detection in cloud computing environment. Applied Soft Computing, 2021. 100: p. 106997.