Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics e-ISSN: 2656-8624
Homepage: https://ijeeemi.org/; Vol. 7, No. 2, pp. 188-202, May 2025

RESEARCH PAPER

OPEN ACCES

Enhancing Software Defect Prediction: HHO-Based Wrapper Feature Selection With Ensemble Methods

Achmad Fauzan Luthfi[1], Rudy Herteno[1], Friska Abadi[1], Radityo Adi Nugroho[1], Muhammad Itqan Mazdadi[1], and Vijay Anant Athavale[2]

[1] Department of Computer Science, Lambung Mangkurat University, Kalimantan Selatan, Indonesia
[2] Department of Computer Science and Engineering, Walchand Institute of Technology, Solapur, Maharashtra, India

ABSTRACT

The growing complexity of data across domains highlights the need for effective classification models capable of addressing issues such as class imbalance and feature redundancy. The NASA MDP dataset poses such challenges due to its diverse characteristics and highly imbalanced classes, which can significantly affect model accuracy. This study proposes a robust classification framework integrating advanced preprocessing, optimization-based feature selection, and ensemble learning techniques to enhance predictive performance. The preprocessing phase involved z-score standardization and robust scaling to normalize data while reducing the impact of outliers. To address class imbalance, the ADASYN technique was employed. Feature selection was performed using Binary Harris Hawk Optimization (BHHO), with K-Nearest Neighbor (KNN) used as an evaluator to determine the most relevant features. Classification models including Random Forest (RF), Support Vector Machine (SVM), and Stacking were evaluated using performance metrics such as accuracy, AUC, precision, recall, and F1-measure. Experimental results indicated that the Stacking model achieved superior performance in several datasets, with the MC1 dataset yielding an accuracy of 0.998 and an AUC of 1.000. However, statistical significance testing revealed that not all observed improvements were meaningful; for example, Stacking significantly outperformed SVM but did not show a significant difference when compared to RF in terms of AUC. This underlines the importance of aligning model choice with dataset characteristics. In conclusion, the integration of advanced preprocessing and metaheuristic optimization contributes positively to software defect prediction. Future research should consider more diverse datasets, alternative optimization techniques, and explainable AI to further enhance model reliability and interpretability.

PAPER HISTORY

Received Feb. 10, 2025
Accepted April 11, 2025
Published April 23, 2025

KEYWORDS

Harris Hawk Optimization;
Feature Selection;
Ensemble Methods;
Preprocessing;
Stacking

CONTACT:

2111016210004@mhs.ulm.ac.id
rudy.herteno@ulm.ac.id
friska.abadi@ulm.ac.id
radityo.adi@ulm.ac.id
mazdadi@ulm.ac.id

1. INTRODUCTION

The quality of software is the most critical aspect of software development. It refers to how well a software program meets the needs or expectations of users, whether explicitly stated or implied. This has a significant impact on businesses, as it can either strengthen or harm a company's brand image [1]. The distribution of defects across software modules can vary significantly. Consequently, applying the same testing effort to all modules within a software project may result in excessive costs and suboptimal outcomes [2]. Software fault prediction represents a systematic methodology that incorporates a range of framework, techniques, and assessment criteria [3]. To address the task of software defect prediction, researchers have employed statistical analysis, machine learning techniques, and, more recently, deep learning approaches [4]. Identifying defective modules is a critical step in test planning. This necessity has driven the development of automated software defect prediction (SDP) processes that leverage metrics derived from historical data. As a result, defect prediction using machine learning techniques has become a prominent research focus, aiming to reduce manual effort in identifying various types of defects in software applications [5]. Early defect prediction enables timely rectification, contributing to the delivery of maintainable software. It allows managers to allocate testing resources effectively, developers to focus on

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

auditing defect-prone code, and testers to prioritize their efforts and resources based on defect-proneness data [6].

The performance of predicting software issues is influenced through how defective data features are represented [7]. Consequently, it is crucial to eliminate non-essential features during the development of the software framework, as these features can introduce noise, increase computational complexity, and reduce the overall accuracy of the model [8]. Feature selection aims to improve the accuracy of Software Defect Prediction (SDP) models by eliminating irrelevant features, thereby reducing the computational complexity and execution time of these algorithms. The three primary approaches to feature selection are the wrapper method, the embedded method, and the filter method. The filter technique evaluates and assigns a score to each feature in the dataset. In contrast, the wrapper technique utilizes classifiers to assess the outcomes of feature selection [9]. Among the different methodologies, wrapper-based techniques stand out as the most widely adopted. Extensive research has revealed that these techniques can effectively minimize the number of features while improving diagnostic accuracy. However, they are not without their difficulties. Typically, these methods utilize heuristic algorithms for feature selection (FS), which can lead to increased computational demands. Moreover, heuristic algorithms often exhibit sensitivity to their parameter settings, which can cause fluctuations in performance. To address these issues, a new approach known as the Binary Harris Hawks Optimization (BHHO) algorithm has been developed. [10].

Harris Hawks Optimization (HHO) is a recently developed swarm-based algorithm and has gained significant popularity in recent years. It simulates the cooperative hunting behavior of Harris hawks to solve optimization problems effectively [11]. Among the mentioned algorithms, the HHO algorithm is a novel metaheuristic approach developed by Heidari et al. [12] in 2019, inspired by the cooperative hunting behavior of sky predators. Due to its effective simulation of the hawks' chasing strategies, this method demonstrates highly competitive performance on certain optimization problems.

Each time a new subset of features is selected, it is used to train the model. The trained model is then tested on a separate test set to directly identify the optimal feature subset from all available features in the dataset [13]. A Software Defect Prediction (SDP) model generally consists of three main components: machine learning algorithms to process and analyze data, soft computing techniques to address uncertainty and complex relationships, and software metrics that represent measurable attributes of the code. These metrics are used to train the machine learning algorithms, enabling them to detect patterns and predict potential defect-prone areas in the code [14]. The process of building a metrics model for software involves collecting metric features to predict defects. However, this approach is often ineffective for projects or versions that vary significantly. To address this limitation, researchers have introduced change metrics to improve the accuracy of defect prediction. Despite its advantages, this technique is less suitable for complex systems in large-scale industries, as it tends to be time-consuming and inefficient. Predicting defects early in the software implementation process helps reduce both implementation and computation costs [7-10].

Forecasting software defects is crucial in the process of Software Development by identifying modules that need thorough testing. Machine learning (ML) techniques, particularly supervised and unsupervised learning, are widely applied for prediction, along with other approaches like semi-supervised and reinforcement learning [15]. Among these, classification methods are the most commonly used for software defect prediction [16]. To improve accuracy, these methods are often integrated with feature selection processes, which focus on identifying the most relevant features while removing those that negatively impact performance.

Researchers have highlighted the issue of class imbalance adversely impacts the forecasting accuracy of software fault prediction models. This situation arises when there is an imbalance in the dataset, characterized by a notable difference in the quantity of data between the majority and minority classes [17]. In many cases, class imbalance leads to overfitting, rendering these models unreliable. To mitigate this issue, researchers have proposed primary solutions such as data selection, combined techniques, and expense-sensitive assessment [18]. Researchers have explored various approaches to solve the class disparity issue. Singh, Misra, and Sharma [19] carried out research on automating bug severity prediction through summaries derived from bug metrics. To handle the inherent class imbalance in the generated bug dataset, they employed ensemble methods, specifically Voting and Bagging. Their findings demonstrated that ensemble methods outperformed single classifiers, indicating that ensemble approaches are effective in dealing with class imbalance issues. The ensemble learning model is constructed by combining several machine learning classifiers to enhance predictive performance [20]. A substantial body of empirical research conducted over the past decade indicates that ensemble methods consistently achieve higher classification accuracy than their individual classifier counterparts [21].

The previous study by Mohammad et al. [14], the authors developed a multi-objective hybrid approach that integrates Hawk Optimization with adaptive synthetic sampling techniques. This model was assessed using

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

various performance metrics, including Area Under the Curve (AUC), precision, recall, and F-measure. Notably, when applied to a healthcare dataset, the model achieved an impressive AUC score of 0.992 and a classification accuracy of 0.99, demonstrating its effectiveness in addressing the challenges of defect prediction in healthcare systems. The primary objective of this research is to identify pertinent features that can effectively predict software defects within healthcare systems, alongside exploring machine learning algorithms that enhance the accuracy of software defect prediction (SDP). The main contributions of this study are outlined as follows:

1. The proposed methodology demonstrates competitive accuracy in Software Defect Prediction by combining advanced preprocessing, metaheuristic-based feature selection using Binary Harris Hawk Optimization (BHHO), and ensemble learning techniques.
2. The application of BHHO for feature selection enables the identification of relevant features while reducing redundancy, which contributes to improving the model's ability to generalize and reducing overfitting risks across diverse datasets.
3. The framework's effectiveness across multiple NASA MDP datasets shows its potential for enhancing defect prediction tasks. However, findings also emphasize that model selection and performance vary with dataset characteristics, and should be validated with statistical significance testing.

This study is organized as follows: Section II provides an overview of the dataset utilized and the proposed methodologies. Section III presents the results of the proposed methods, focusing on accuracy and AUC, along with optimization parameters such as precision, recall, and F1-measure. Section IV offers an interpretation of the results, comparing them with findings from other studies while also addressing any limitations encountered. Finally, Section V concludes the study by summarizing the objectives, key findings, and suggestions for future research directions.

2. MATERIALS AND METHOD

The proposed method in this research begins with the utilization of the NASA MDP dataset. Initially, preprocessing is conducted, which includes data transformation through z-score standardization, ensuring that all values have a mean of 0 and a standard deviation of 1. This step reduces the influence of dimensionality and plays a crucial role in outlier detection, as Z-scores significantly distant from 0 may indicate the presence of outliers. Following this, scaling techniques such as robust scaling are applied to center the data around zero and adjust it to a more consistent scale, thereby minimizing the impact of extreme values on the results. Subsequently, sampling techniques like ADASYN are employed to address class imbalance, enhancing the classifier's performance. Then, Feature selection performed using a wrapper method, specifically employing binary Harris Hawk Optimization (BHHO) with KNN as the evaluator for feature selection. The log loss function is utilized as the objective function in the binary HHO process. The model is subsequently tested using various classification and ensemble algorithms, including Random Forest (RF), Support Vector Machine (SVM), and Stacking. Finally, the model's performance is evaluated using metrics such as Area Under the Curve (AUC) and accuracy. Fig. 1 illustrates the research flow adopted in this study.

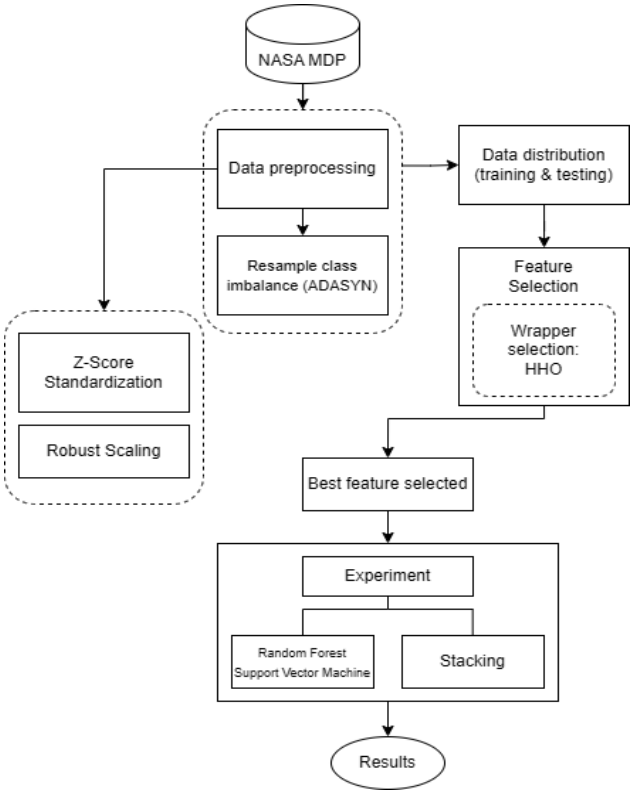


Fig. 1. Flowchart illustrating the research methodology based on the proposed approach.

A. Dataset

The prediction of software defects is conducted by leveraging datasets from previous software projects [22]. One of the most frequently utilized datasets is the National Aeronautics and Space Administration (NASA) dataset, which is well-known and widely used in the development of defect prediction models due to its public accessibility. In total, dataset used for NASA benchmarks comprises 13 distinct software projects, each representing a specific software system or subsystem developed by NASA. This dataset includes a wide range of instances, from 127 to 9,277, and features 20 to 40 metrics that focus on identifying defects and analyzing static code. Furthermore, a bug tracking system is utilized to keep

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

track of the number of errors for each instance. The quality of the software is closely tied to key static code metrics, which encompass aspects such as size, readability, and complexity. However, this dataset presents challenges related to data imbalance, where the quantity of non-defective instances greatly exceeds that of the faulty ones, along with those presence of noise disturbances. This imbalance poses a common challenge, as prediction outcomes tend to be biased towards the non-defective class. To address this issue, it is essential to adjust prediction techniques by integrating or combining other algorithms [23].

Two class labels from NASA MDP dataset comprises: “Y” (representing defective) and “N” (representing non-defective). In the preprocessing phase, these categorical labels were transformed into numeric values, where Y is encoded as 1 and N is encoded as 0 [24]. This conversion facilitates the application of various machine learning algorithms that require numerical input for effective processing and analysis. The version of the NASA MDP dataset utilized in this research is D'', sourced from (https://github.com/klainfo/NASADefectDataset). The data volume for each dataset is presented in Table 1.

Table 1. Detailed information regarding the NASA MDP Dataset, including its characteristics and relevant metrics.

Dataset	Number of features	Sample sizes	Number of Defects	Defects (%)
CM1	38	327	42	12.8
JM1	22	7720	1612	20.8
KC1	22	1162	294	25.3
KC3	40	194	36	18.5
MC1	38	1988	46	2.3
MC2	40	124	44	35.4
MW1	38	253	27	10.7
PC1	38	705	61	9.7
PC2	37	722	16	2.2
PC3	38	1077	134	12.4
PC4	38	1270	176	13.8
PC5	39	1694	458	27.0

In this study, the labels in the MC1 dataset were transformed, with the designation "Y" converted to "1" and "N" converted to "0." This modification is illustrated in Table 2 and Table 3.

B. Data Transformation

This process is crucial to ensure that the data used in the model is high quality, thereby enabling accurate results. By applying appropriate transformations, the impact of outliers can be minimized, distributions can be normalized, and the model's performance in predicting defects can be effectively enhanced [25]. Before preprocessing, the data was split in a 70:30 ratio, with 70% allocated for training and 30% for testing, using random_state=42 to ensure reproducibility. Then the technique standardizes feature values to have a mean of 0 and a standard deviation of 1 using the formula as shown in Eq. (1).

$$X_{new} = \frac{X - \bar{X}}{\sigma} \quad (1)$$

X_{new} represents the standardized value indicating how far the original value X is from the mean $\bar{X}$ in terms of standard deviation σ. This is achieved by subtracting the mean from the original value and dividing by the standard deviation, resulting in a value that reflects the relative deviation from the mean. This transformation is crucial for standardizing data, ensuring a mean of 0 and a standard deviation of 1, and is effective for detecting outliers, as Z values far from 0 may indicate their presence. In the implementation, Z-scores were calculated using the stats.zscore function from the SciPy library. To identify and handle outliers, a threshold of 2 was set, meaning any data point with a Z-score greater than this threshold is considered an outlier. Consequently, an outlier mask was created to filter out these points, ensuring that only data within the threshold is retained for training. This approach is used due to its effectiveness in standardizing the dataset, minimizing the impact of outliers, and ensuring that the features are on a comparable scale, which ultimately enhances the model's predictive performance.

C. Scalling Techniques

The performance of machine learning models is influenced by various factors, including the scale of features in the dataset. Feature scale differences can cause inaccuracies by favoring larger values.

Table 2. Overview of the CM1 Dataset Labels Before Preprocessing: Original Categorical Labels and Their Corresponding Numeric Transformations

id	LOC_BLANK	BRANCH_COUNT	…	NUM_UNIQUE_OPERATORS	LOC_TOTAL	label
0	3	1	…	5	17	N
1	3	3	…	14	14	N
2	1	5	…	9	13	N
…	…	…	…	…	…	…
7718	9	35	…	38	60	N
7719	3	15	…	17	30	N

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

Table 3. Overview of the CM1 Dataset Labels After Preprocessing: Numeric Representations of Categorical Labels Following Transformation

id	LOC_BLANK	BRANCH_COUNT	…	NUM_UNIQUE_OPERATORS	LOC_TOTAL	label
0	3	1	…	5	17	0
1	3	3	…	14	14	0
2	1	5	…	9	13	0
…	…	…	…	…	…	…
7718	9	35	…	38	60	0
7719	3	15	…	17	30	0

To address this, scaling techniques are employed to ensure all features have a uniform range of values [26]. The Robust Scaler minimize the impact of outliers by centering data around the median (the second quartile, Q2(x)) and scaling based on the interquartile range. The interquartile range is the difference between the first quartile Q1(x) and the third quartile Q3(x), shown in the Eq. (2).

$$x'_i = \frac{x_i - Q_2(x)}{Q_3(x) - Q_1(x)} \quad (2)$$

This method standardizes x'_i by subtracting the median $Q_2(x)$ from the original value xi and dividing by the interquartile range. This centers the data around the median and adjusts the scale to be more consistent, reducing the influence of extreme values. ppp In the implementation, the RobustScaler from the Scikit-learn library was initialized with the following parameters: with_centering set to True to center the data around the median, with_scaling set to True to scale the data to the interquartile range, and the quantile_range set to the default of (25.0, 75.0) to calculate the IQR. The copy parameter was set to True, ensuring that a copy of the data is created during scaling, while unit_variance was set to False, meaning the data would not be scaled to achieve a variance of 1 for normally distributed features. This approach is used due to its ability to minimize the impact of outliers by centering the data around the median and scaling based on the interquartile range, ensuring that the model remains robust and accurate even in the presence of extreme values.

D. Sampling Techniques

The performance of classifiers is influenced by various factors, including the number of samples and the types of classes analyzed. Data imbalance occurs when the minority class has significantly fewer instances than the majority class, leading to challenges for machine learning-based classifiers and diminishing their overall effectiveness. Numerous studies have explored this issue and proposed solutions, with Bowyer identifying two primary approaches: data-based and algorithm-based methods [9]. The data-based approach focuses on balancing class distribution through resampling techniques, which can involve over-sampling the minority class or under-sampling the majority class, either randomly or systematically. Various methods have been suggested to address imbalanced data, including ADASYN, SMOTE, and Random Oversampling [27]. Adaptive Synthetic Sampling (ADASYN) is a method aimed at addressing dataset imbalance to improve classifier performance (Algorithm 1). It synthesizes minority class samples based on their distribution in the training dataset, focusing more on difficult-to-learn samples and less on easier ones. The approach determines a probability distribution to generate additional minority class samples, resulting in a more balanced dataset [14]. This mechanism ensures that the distribution of new samples reflects the existing data patterns, particularly in the minority class, thereby enhancing class balance without losing important information. The parameters used in the ADASYN implementation include sampling_strategy ='auto', which automatically targets the majority class for resampling, ensuring a balanced class distribution. Additionally, random_state=42 is set to control the randomization of the algorithm, allowing for reproducible results. The n_neighbors parameter is set to 5, indicating that five nearest neighbors will be used to define the neighborhood of samples for generating synthetic samples. This configuration is chosen to enhance the model's ability to learn from the minority class, thereby improving overall classifier performance while maintaining the integrity of the data distribution. The following provides a more detailed explanation of how the ADASYN technique is implemented in the pseudocode presented.

E. Feature Selection

In classification, techniques for feature selection are primarily grouped into three categories: filter methods, wrapper methods, and embedded methods [28]. Key differences include whether selection is performed separately or integrated with the learning algorithm, the evaluation metrics used, computational complexity, and the ability to detect redundancy and feature interactions. Filter methods evaluate features based on their relationships and are generally faster, while wrapper methods use learning algorithms for evaluation and tend

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

to be more accurate [29]. Feature selection (FS) algorithms are divided into two main categories: exact algorithms and meta-heuristic search algorithms [9]. Meta-heuristic algorithms typically outperform exact methods, especially for complex problems. They are further classified into single-solution algorithms, such as Tabu Search [30], which emphasize exploitation, and population-based algorithms, like Harris Hawks Optimization (HHO) [31], which focus on exploration. While population-based algorithms explore broader search areas and yield more accurate results, single-solution algorithms generally execute faster. Binary encoding is a simple approach for FS, where 0 indicates a feature is not selected and 1 indicates it is selected. The application of feature selection (FS) algorithms will select m features from the original n features, where m ≤ n.

Algorithm1. Adaptive Synthetic

Input:
- Dataset S1 with n samples vi and labels wi (0 for minority class, 1 for majority class).

Output:
- New synthetic data.

Steps:
1. Calculate Class Imbalance:
   - Find the ratio of minority to majority samples.
2. Determine Total Samples to Synthesize:
   - Calculate S = (n1 - n2) * β, where n1 is the number of minority samples, n2 is the number of majority samples, and β is a coefficient.
3. For Each Minority Sample vi:
   - Identify the K nearest neighbors.
   - Calculate the ratio ri = Δi / K, where Δi is the number of observations in the K-nearest neighbors.
   - Normalize ri to create a probability distribution: ri = ri / Σri.
   - Compute si = ri * S, indicating the total synthetic samples needed for each minority sample vi.
4. Generate Synthetic Samples:
   - For m = 1, 2, …, si:
     - Randomly select a sample vm from the K-nearest neighbors of vi.
     - Create a synthetic sample vq using the formula: vq = vi + (vi - vm).
5. End of Algorithm.

The Harris Hawks Optimizer (HHO) was selected over other metaheuristic algorithms due to its innovative design that incorporates two distinct exploration phases and four exploitation phases, enhancing its ability to navigate complex search spaces effectively. While numerous well-established algorithms such as the Whale Optimization Algorithm (WOA) [32], Dragonfly Algorithm (DA) [33], Grasshopper Optimization Algorithm (GOA) [34], Grey Wolf Optimizer (GWO) [35], Multi-Verse Optimizer (MVO) [36], and Moth-Flame Optimizer (MFO) [37] demonstrate robust search capabilities, HHO's unique structure allows for a more refined balance between exploration and exploitation. This capability enables HHO to achieve superior performance in solving diverse mathematical and engineering problems, making it a compelling choice for applications requiring high-quality solutions within reasonable computational timeframes.

F. Harris Hawk Optimization

Harris Hawk Optimization (HHO) is a novel optimization algorithm inspired by the hunting dynamics between hawks and rabbits. Its mathematical foundation allows it to effectively tackle various constrained and unconstrained problems. The algorithm updates search agents through two exploration phases and four exploitation phases [38]. Harris Hawk Optimization (HHO) was initially designed to operate in continuous search spaces. However, real-world problems like feature selection require binary search spaces. To address this, the algorithm has been efficiently reformulated to function in binary spaces. In several studies, a two-phase binarization technique has been employed to present a binary variant of HHO, referred to as Binary Harris Hawk Optimization (BHHO) [39]. Wrapper-based feature selection using Binary Harris Hawk Optimization (BHHO) is combined with the K-Nearest Neighbor (KNN) classifier as the evaluator with default parameters (n_neighbors=5) as the evaluator for HHO. KNN is selected for its simplicity and common use as a non-parametric classification technique, and it has shown competitive performance in various studies compared to other feature selection methods [38]. BHHO is specifically designed to handle binary problems, making it ideal for scenarios where the solution requires a binary decision, such as selecting or rejecting features. When adapting metaheuristic algorithms for optimization, two key aspects are crucial: solution representation and the objective function.

Solution Representation: In feature selection (FS), each feature can be either selected or not, represented as a binary vector X={x1,x2,…,xN}, total count of features represent with N.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

Target function: Guiding the search process is crucial by assessing candidate solutions according to their quality. The main aim of feature selection (FS) is to minimize the count of chosen features while improving classification effectiveness. This research, a single-target HHO algorithm is utilized, with the objective function specified as Log Loss, as illustrated in Eq. (3).

$$Log\ Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y)\log(1-p_i)] \quad (3)$$

N represents the total number of samples, y_i is the actual class label (0 or 1), and p_i is the predicted probability of the class. Log Loss measures the performance of the model, providing a lower value for better classification accuracy. Python library for performing feature selection using a variety of nature inspired wrapper algorithms is taken from (https://github.com/jaswinder9051998/zoofs). In this research, the parameters proposed by the HHO algorithm, specifically a population size of 10 and 30 iterations by previous analyses [14] have indicated that this combination yields the optimal performance for the predictive model.

G. Classification

Random Forest (RF) and Support Vector Machine (SVM) are the two key classifiers employed to differentiate between software modules with defects and those without. These methodologies are integral to machine learning (ML) and demonstrate substantial effectiveness in classification performance. Their primary goal is to identify patterns that indicate specific classes, connecting each data instance to the existing dataset [40].

1. Random Forest

Random Forest (RF) is an ensemble method that consists of multiple decision trees, combining their predictions to improve accuracy and robustness [41]. It effectively handles datasets with many irrelevant attributes by selecting the most informative ones for classification. RF constructs decision trees by randomly sampling data subsets and mapping them to randomly chosen feature subspaces [42]. Random Forest is represented in Eq. (4).

$$\{DT(x,\theta_k)\}_{k=1}^{T} \quad (4)$$

In this equation, the k-th decision tree is trained with input data x and random parameters θ_k, which include a subset of data from bagging and a random subset of features at each node. By building T independent decision trees, predictions are combined using majority voting for classification or averaging for regression, improving model accuracy and reducing the risk of overfitting. In this Random Forest implementation, the parameters used are n_estimators=1000 [43], which specifies the number of trees in the forest to enhance predictive accuracy. Additionally, max_depth=None allows the trees to grow until all leaves are pure, ensuring that the model captures the complexity of the data. The parameter min_samples_leaf=1 is also set, which specifies the minimum number of samples required to be at a leaf node. This configuration is designed to optimize the model's performance by balancing accuracy and robustness.

2. Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks [44]. SVM creates an optimal hyperplane to efficiently differentiate data sample into separate classes by transforming them into an expanded-dimensional space. This method is especially effective for two-class classification applications with significant dimensionality, rendering it useful for applications such as Software Defect Prediction (SDP). SVM's ability to handle complex, non-linear data and maximize the margin between classes enhances its generalization capabilities, contributing to its widespread use in fields such as image processing and bioinformatics. Support Vector Classification (SVC) is a classification method that utilizes the Support Vector Machine (SVM) algorithm to separate data into classes. It can employ various kernels, including linear, polynomial, and radial basis function (RBF) kernels. RBF SVC is particularly effective for non-linearly separable data, as it measures the distance between data points and class centers, allowing for complex decision boundaries. Its strengths include handling intricate patterns, flexibility in adjusting parameters γ and C, and strong performance in applications such as text, image, and medical data analysis. The RBF kernel equation is represented in Eq. (5).

$$K(x,y) = e^{-\gamma\|x-y\|^2} \quad (5)$$

γ is a parameter that controls the influence of a single training example. A small value of γ results in smoother decision boundaries, while a larger value leads to sharper decision boundaries. This parameter plays a crucial role in determining the model's flexibility and its ability to generalize from the training data. In the implementation we used the default parameters used include C=1.0, which serves as the regularization parameter, controlling the trade-off between achieving a low training error and a low testing error. The kernel='rbf' is specified as the kernel type, which is particularly effective for handling non-linear relationships in the data. Additionally, random_state=42 is set to ensure reproducibility of results by controlling the randomness in the algorithm. The probability=True parameter is also included, allowing the model to output probability estimates for the predictions. This configuration is designed to optimize the model's performance while providing reliable probability estimates for classification tasks.

3. Stacking

Stacking is a heterogeneous ensemble model that combines multiple base classifiers through a meta-classifier to produce a final prediction model [45]. The base classifiers use different learning algorithms and are trained on the entire dataset, while their outputs serve as the training data for the meta-classifier to build the final model. In the implementation, the estimators parameter is

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

defined as a list of tuples, where each tuple consists of a string (name) and an estimator instance. For example, the base estimators include a RandomForestClassifier with random_state=42 and an SVC with probability=True and random_state=42. This combination allows for diverse learning algorithms to be stacked together. The final_estimator parameter is set to a classifier that will be used to combine the outputs of the base estimators, effectively creating a robust ensemble model. This configuration enhances the model's predictive performance by leveraging the strengths of multiple classifiers.

H. Evaluation Metrics

Assessment metrics such as accuracy, specificity, sensitivity, precision, and ROC-AUC are integral to prediction and classification. This study focuses on analyzing the suggested framework through accuracy and AUC values, utilizing ROC-AUC, which is frequently applied in the context of software defect prediction. AUC useful in class imbalance situations, providing a comprehensive view of model performance across thresholds. The calculation of AUC is based on the ratio of false positive rates to true positive rates. Accuracy serves as a measurement that effectively differentiates between defective and non-defective software components. It is computed as the ratio of true positives to true negatives across all instances, as illustrated in Eq. (6).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (6)$$

TP (True Positive) and TN (True Negative) represent correct predictions for positive and negative classes, while FP (False Positive) and FN (False Negative) indicate incorrect predictions. Accuracy measures the model's effectiveness in classifying data, ranging from 0 to 1, with 1 indicating perfect predictions. AUC (Area Under the Curve) assesses the model's capability to distinguish between true positive rates and false positive rates across different thresholds. A higher AUC value indicates better model performance in distinguishing between classes [35]. The AUC is represented in Eq. (7).

$$AUC = \frac{1 + TPR - FPR}{2} \quad (7)$$

True Positive Rate (TPR) measures the proportion of correct positive predictions, while False Positive Rate (FPR) indicates incorrect negative predictions. AUC ranges from 0 to 1, with 1 representing perfect performance and 0.5 indicating random predictions. Key optimization parameters include precision, recall, and F1-measure. The F1 Score is selected as an evaluation metric because it effectively balances precision and recall, making it suitable for imbalanced datasets. Precision measures the accuracy of positive predictions, while recall indicates the model's ability to identify all relevant positive instances. Unlike accuracy, the F1 Score accounts for both false positives and false negatives, providing a clearer picture of model performance [46]. Furthermore, in the calculations for AUC and F1-score, precision reflects how accurately positive instances are identified among all predicted positives, highlighting the relevance of using the F1 Score in contexts with uneven class distributions.

3. RESULTS

This section presents the results of the proposed framework in a clear and organized manner, using tables and figures to effectively illustrate key findings. The evaluation was conducted using the NASA MDP datasets with the proposed framework implemented in Jupyter Notebook and tested on a machine with a Ryzen 5 4600H CPU @ 3.00GHz and 16 GB DDR4 RAM. Performance was measured using accuracy, precision, recall, F1-score, and AUC metrics.

To statistically validate the observed differences in model performance, this study employs independent samples t-tests. The primary purpose of the t-test is to determine whether the differences in performance metrics—particularly accuracy and AUC—between classifiers are statistically significant or could have occurred by chance. This allows for a more rigorous and objective evaluation of model effectiveness beyond simple numerical comparisons. By applying this test, the study avoids overclaiming performance advantages and strengthens the reliability of its conclusions.

A. Selected Features Using BHHO

Table 4 displays the number of features selected by the Binary Harris Hawk Optimization (BHHO) from each dataset. These selected features are considered most relevant to defect prediction.

Table 4. Displays the features that have been selected for analysis, highlighting their significance and relevance to the study.

Dataset	Features	Feature Selection BHHO
CM1	38	26
JM1	22	10
KC1	22	14
KC3	40	26
MC1	38	26
MC2	40	22
MW1	38	21
PC1	38	21
PC2	37	22
PC3	38	25
PC4	38	25
PC5	39	30

The features repeatedly selected across datasets include metrics such as CYCLOMATIC_COMPLEXITY,

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

HALSTEAD_EFFORT, HALSTEAD_ERROR_EST, LOC_COMMENTS, and LOC_TOTAL. Their frequent occurrence indicates their relevance in predicting software defects. For instance, the presence of LOC_COMMENTS suggests the influence of code documentation on software quality.

B. Accuracy and AUC Performance

Table 5 provides accuracy and AUC values for three models: Random Forest (RF), Support Vector Machine (SVM), and Stacking. These models were applied after the BHHO-based feature selection process.

Table 5. Results of various classification algorithms based on the proposed approach are presented, showcasing their performance and effectiveness in the analysis.

Data		RF	SVM	Stacking
CM1	Accuracy	0.961	0.868	0.961
	AUC	0.982	0.979	0.987
JM1	Accuracy	0.874	0.603	0.875
	AUC	0.942	0.648	0.940
KC1	Accuracy	0.834	0.629	0.837
	AUC	0.894	0.675	0.893
KC3	Accuracy	0.902	0.902	0.927
	AUC	0.974	0.988	0.990
MC1	Accuracy	0.966	0.981	0.998
	AUC	1.000	0.998	1.000
MC2	Accuracy	0.952	0.810	0.857
	AUC	0.955	0.891	0.945
MW1	Accuracy	0.966	0.862	0.931
	AUC	0.996	0.926	0.987
PC1	Accuracy	0.976	0.940	0.964
	AUC	0.994	0.990	0.992
PC2	Accuracy	0.956	0.962	0.978
	AUC	0.997	0.991	0.999
PC3	Accuracy	0.952	0.865	0.961
	AUC	0.994	0.952	0.994
PC4	Accuracy	0.915	0.872	0.918
	AUC	0.970	0.935	0.973
PC5	Accuracy	0.816	0.749	0.810
	AUC	0.907	0.821	0.902

Stacking and RF consistently achieved the highest performance. SVM performed notably poorer in datasets such as JM1, KC1, and PC5, possibly due to suboptimal parameters or kernel choice. Adjusting model parameters through grid search or random search may help identify optimal SVM parameters.

C. Evaluation of Precision, Recall, and F1-Measure

Table 6 presents the precision, recall, and F1-measure of the three classifiers. These metrics further confirm the effectiveness of the proposed method with selected features.

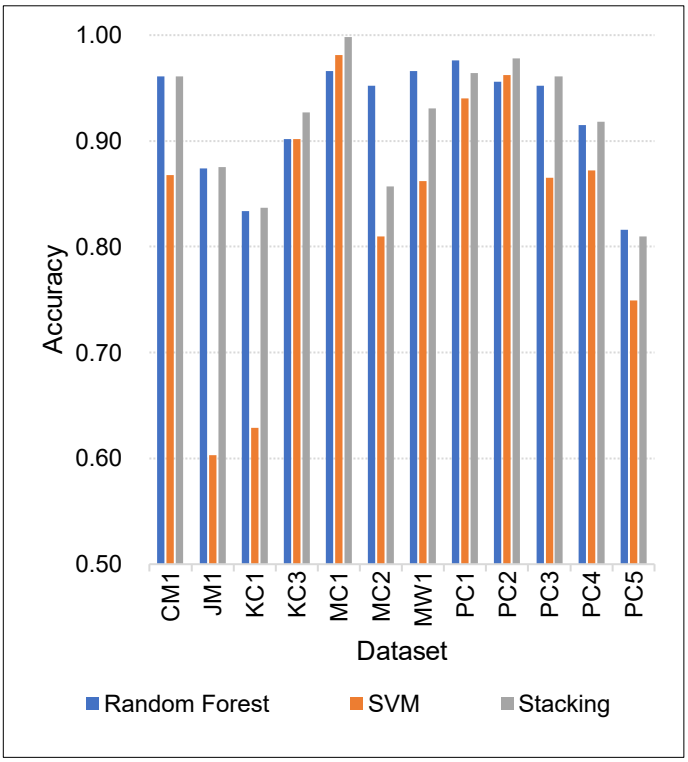
Table 6. The effectiveness and performance of the optimization parameters were thoroughly evaluated and analyzed in this study.

Data		Precision	Recall	F1-measure
CM1	RF	0.952	0.976	0.964
	SVM	0.804	1	0.891
	Stacking	0.952	0.976	0.964
JM1	RF	0.901	0.855	0.877
	SVM	0.639	0.560	0.597
	Stacking	0.900	0.857	0.878
KC1	RF	0.860	0.806	0.832
	SVM	0.634	0.650	0.642
	Stacking	0.861	0.812	0.836
KC3	RF	1	0.810	0.895
	SVM	0.905	0.905	0.905
	Stacking	0.95	0.905	0.927
MC1	RF	1	0.993	0.996
	SVM	0.965	1	0.982
	Stacking	0.996	1	0.998
MC2	RF	1	0.909	0.952
	SVM	0.769	0.909	0.833
	Stacking	0.833	0.909	0.870
MW1	RF	0.966	0.966	0.966
	SVM	0.889	0.828	0.857
	Stacking	0.903	0.966	0.933
PC1	RF	0.974	0.974	0.974
	SVM	0.893	0.987	0.938
	Stacking	0.949	0.974	0.961
PC2	RF	0.940	0.963	0.952
	SVM	0.921	1	0.959
	Stacking	0.953	1	0.976
PC3	RF	0.935	0.962	0.948
	SVM	0.783	0.971	0.867
	Stacking	0.952	0.962	0.957
PC4	RF	0.898	0.936	0.917
	SVM	0.807	0.979	0.885
	Stacking	0.904	0.936	0.920
PC5	RF	0.754	0.885	0.814
	SVM	0.701	0.782	0.739
	Stacking	0.749	0.878	0.808

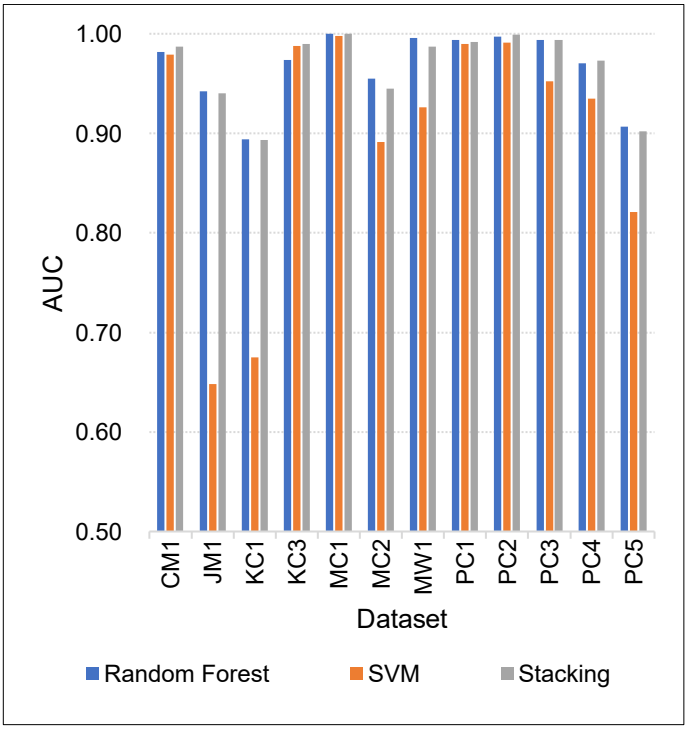
D. Visualization of Results

Fig 2 illustrates the accuracy and AUC across experiments:

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043



(a)



(b)

Fig. 2. Evaluation of Accuracy and AUC outcomes for each experiment: (a) Accuracy metrics for all experiments, (b) AUC metrics for all experiments.

E. Statistical Analysis of Model Performance

Table 7 and Table 8 show the accuracy and AUC scores obtained across experiments for each classification model. To further validate the observed performance differences, a paired t-test was conducted between models.

Table 7. Statistical comparison of accuracy between classification models thoroughly evaluated and analyzed in this study.

Model Comparison	t-Statistic	p-Value	Significant ($p < 0.05$)
RF vs SVM	2.231	0.036	Yes
RF vs Stacking	0.188	0.853	No
SVM vs Stacking	-2.079	0.049	Yes

Table 8. Statistical comparison of AUC between classification models thoroughly evaluated and analyzed in this study.

Model Comparison	t-Statistic	p-Value	Significant ($p < 0.05$)
RF vs SVM	1.834	0.080	No
RF vs Stacking	0.017	0.987	No
SVM vs Stacking	-1.820	0.082	No

The test compared the accuracy and AUC values across the Random Forest (RF), Support Vector Machine (SVM), and Stacking models over twelve datasets. Results showed that, in terms of accuracy, the differences between RF and SVM ($p = 0.036$) and between SVM and Stacking ($p = 0.049$) were statistically significant at the 5% level. These findings indicate that both RF and Stacking significantly outperformed SVM. However, no significant difference was found between RF and Stacking ($p = 0.853$), suggesting their accuracies were statistically comparable. Regarding AUC, while RF and Stacking generally outperformed SVM across most datasets, the differences were not statistically significant. The comparison between RF and SVM yielded a p-value of 0.080, while that of SVM and Stacking was $p = 0.082$, both exceeding the typical 0.05 threshold.

This implies that although trends favored ensemble methods, the variation in AUC values may not be strong enough to conclusively assert superiority based on the available data. These statistical insights reinforce the robustness of the proposed ensemble methods, particularly in terms of accuracy, while also cautioning against over-claims regarding AUC differences without stronger statistical backing.

4. DISCUSSION

The feature selection results demonstrate that the Binary Harris Hawks Optimization (BHHO) method not only helps in reducing model complexity but also contributes to

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

improving classification accuracy. Compared to prior studies using healthcare datasets, the proposed method exhibits greater efficiency in identifying relevant features. This directly supports the enhancement of predictive performance. These findings align well with the research objectives, demonstrating how selected features influence classification outcomes. Furthermore, the consistency of these results with existing literature underlines the robustness of the BHHO method.

Table 9. Comparison of performance outcomes from previous studies

Dataset	Model	AUC	Accuracy
Healthcare [14]	Adasyn + HHO	0.992	0.990
12 Nasa MDP Datasets (Our)	Z-transformation + Robust Scaler + Adasyn + HHO	0.998	1.000

As shown in Table 9, our method shows higher AUC and accuracy than the referenced healthcare dataset. While the healthcare dataset using ADASYN and HHO reached an AUC of 0.992 and accuracy of 0.990, our approach achieved 0.998 and accuracy of 1.000 on the NASA MDP datasets. Although this suggests a potential performance improvement, caution should be exercised in generalizing the results due to differences in datasets and experimental conditions. Among the classifiers, Random Forest (RF) consistently achieved high accuracy and F1 scores, reaching a maximum F1 of 0.996 on the MC1 dataset. Stacking also maintained competitive performance, while Support Vector Machine (SVM) showed high variability, particularly on JM1 and KC1 where its F1-score fell below 0.600. This variability suggests that SVM may require more careful tuning and may not generalize as effectively as ensemble methods in certain contexts.

The use of t-tests further supports the interpretation of these results. The test confirms statistically significant differences in accuracy between SVM and ensemble methods such as RF and Stacking, while differences in AUC were not statistically significant. These findings validate the reliability of our results in terms of accuracy but also emphasize the need for cautious interpretation when comparing AUC values.

SVM's weaker performance on datasets like JM1 and KC1 may be attributed to several factors: sensitivity to undetected outliers despite preprocessing, limitations in selected features, suboptimal kernel or parameter choices, and vulnerability to overfitting or underfitting depending on dataset size and complexity. This reinforces the need for careful parameter tuning and appropriate kernel selection when using SVM in defect prediction.

The proposed methodology contributes to advancing software defect prediction by integrating HHO-based feature selection and ensemble classifiers. Statistically significant improvements in accuracy, and consistent performance across multiple datasets, support the hypothesis that this combination enhances defect prediction. These results validate the framework's potential for handling high-dimensional datasets and improving classifier generalization, within the tested conditions.

A comparison with recent studies further validates the effectiveness of the proposed method. Ali et al. [47] introduced a five-stage framework integrating genetic algorithm-based feature selection with an ensemble of Random Forest, SVM, and Naïve Bayes, followed by majority voting. Their method achieved a maximum accuracy of 95.1% and reduced training and testing time by 51.52% and 52.31%, respectively, on NASA datasets such as CM1, JM1, and PC3. Similarly, Balogun et al. [48] proposed an Enhanced Wrapper-based Feature Selection (EWFS) method and evaluated it using Naïve Bayes and Decision Tree classifiers. Their experiments, conducted over 25 datasets with 10-fold cross-validation repeated 10 times, reported average accuracies of 82.57% (NB) and 83.07% (DT), with corresponding AUC scores of 0.783 and 0.723, and F-measure values of 0.807 and 0.820. Although our method demonstrates strong performance in this context, further studies are needed to validate its generalizability across different datasets and domains.

However, certain limitations must be acknowledged. The preprocessing pipeline including z-score normalization, robust scaling, and ADASYN may not generalize across all datasets. The HHO algorithm, while effective, can be sensitive to parameter settings, and its iterative nature increases computational cost. Additionally, ensemble classifiers, despite their strong performance, require more computational resources. All experiments rely on NASA MDP datasets, which may not fully reflect the diversity of modern software projects. Broader evaluation on real-world industrial datasets is recommended.

From a trade-off perspective, while Stacking offers excellent accuracy and stability, it also introduces greater training complexity compared to simpler models. Likewise, feature selection via HHO enhances prediction but demands more computation. In practical environments, these trade-offs must be considered based on context and available resources. The practical implications are clear: the proposed framework assists in identifying defect-prone modules early in development, improving testing efficiency and software quality. Its adaptability across datasets supports deployment in real-world CI/CD environments, particularly for organizations maintaining historical defect data.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

5. CONCLUSION

The primary objective of this research was to develop a reliable classification model using the NASA MDP dataset by applying a combination of preprocessing techniques, feature selection via Binary Harris Hawks Optimization (BHHO), and classification algorithms such as Random Forest (RF), Support Vector Machine (SVM), and Stacking. The study addressed key challenges in software defect prediction, including class imbalance and high-dimensional feature spaces, and evaluated model performance using accuracy, AUC, precision, recall, and F1-measure.

The findings indicate that the Stacking model achieved competitive performance across many datasets. For example, in the MC1 dataset, Stacking achieved an accuracy of 0.998 and an AUC of 1.000. It also showed high precision (0.996), recall (1.000), and F1-measure (0.998), reflecting strong performance in balancing sensitivity and specificity. However, it is important to interpret these results within the context of the tested datasets and experimental setup. In contrast, the SVM model showed lower performance on several datasets, particularly on JM1, where its F1-measure dropped to 0.597, suggesting potential limitations in handling class imbalance and complex data distributions without optimal parameter tuning.

The Random Forest classifier also demonstrated strong performance, particularly on datasets such as CM1 and MC1, with precision values of 0.952 and 1.000, respectively. However, in datasets like JM1 and KC1, RF's F1-measure was slightly lower than that of Stacking, highlighting the importance of selecting classification models based on the characteristics of individual datasets.

Based on these findings, future research may explore alternative or hybrid metaheuristic algorithms beyond BHHO such as Whale Optimization Algorithm or Firefly Algorithm—to potentially improve convergence and feature selection quality. Expanding evaluations to include larger and more diverse datasets, including industrial systems or open-source repositories (e.g., GitHub, GitLab), would also support broader generalization. Additionally, integrating deep learning approaches like graph neural networks or transformer-based models, and combining static code metrics with process or textual features, may offer richer representations. Finally, incorporating explainable AI (XAI) techniques could enhance transparency and trust in defect prediction models, supporting more informed decision-making in software engineering practice.

REFERENCES

[1] J. Kethireddy, E. Aravind, and M. Vijayakamal, “Software Defects Prediction using Machine Learning Algorithms,” Nov. 2023.

[2] M. Canaparo, E. Ronchieri, and G. Bertaccini, “Software defect prediction: A study on software metrics using statistical and machine learning methods,” Nov. 2022, p. 20. doi: 10.22323/1.415.0020.

[3] M. B. R. Pandit and N. Varma, “A Deep Introduction to AI Based Software Defect Prediction (SDP) and its Current Challenges,” in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 284–290. doi: 10.1109/TENCON.2019.8929661.

[4] S. Omri and C. Sinz, “Deep Learning for Software Defect Prediction: A Survey,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, in ICSEW’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 209–214. doi: 10.1145/3387940.3391463.

[5] S. Haldar and L. Capretz, “Interpretable Software Defect Prediction from Project Effort and Static Code Metrics,” *Computers*, vol. 13, p. 52, Nov. 2024, doi: 10.3390/computers13020052.

[6] R. Malhotra and K. Khan, “A Study on Software Defect Prediction using Feature Extraction Techniques,” in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2020, pp. 1139–1144. doi: 10.1109/ICRITO48877.2020.9197999.

[7] H. Turabieh, M. Mafarja, and X. Li, “Iterated feature selection algorithms with layered recurrent neural network for software fault prediction,” *Expert Syst Appl*, vol. 122, pp. 27–42, 2019, doi: https://doi.org/10.1016/j.eswa.2018.12.033.

[8] P. Kumar, G. Gupta, and R. Tripathi, “Toward Design of an Intelligent Cyber Attack Detection System using Hybrid Feature Reduced Approach for IoT Networks,” *Arab J Sci Eng*, vol. 46, Nov. 2021, doi: 10.1007/s13369-020-05181-3.

[9] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, “Enhanced Binary Moth Flame Optimization as a Feature Selection Algorithm to Predict Software Fault Prediction,” *IEEE Access*, vol. 8, pp. 8041–8055, 2020, doi: 10.1109/ACCESS.2020.2964321.

[10] M. Dong, Y. Wang, Y. Todo, and Y. Hua, “A Novel Feature Selection Strategy Based on the Harris Hawks Optimization Algorithm for the Diagnosis of Cervical Cancer,” *Electronics (Basel)*, vol. 13, no. 13, 2024, doi: 10.3390/electronics13132554.

[11] S. Song *et al.*, “Dimension decided Harris hawks optimization with Gaussian mutation: Balance analysis and diversity patterns,” *Knowl Based Syst*, vol. 215, p. 106425, 2021, doi: https://doi.org/10.1016/j.knosys.2020.106425.

[12] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, “Harris hawks optimization:

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

Algorithm and applications," *Future Generation Computer Systems*, vol. 97, pp. 849–872, 2019, doi: https://doi.org/10.1016/j.future.2019.02.028.

[13] L. Peng, Z.-N. Cai, A. A. Heidari, L. Zhang, and H. Chen, "Hierarchical Harris hawks optimizer for feature selection (Journal of Advanced Research, Impact factor 12.822)," *J Adv Res*, Nov. 2023, doi: 10.1016/j.jare.2023.01.014.

[14] U. G. Mohammad, S. Imtiaz, M. Shakya, A. Almadhor, and F. Anwar, "An Optimized Feature Selection Method Using Ensemble Classifiers in Software Defect Prediction for Healthcare Systems," *Wirel Commun Mob Comput*, vol. 2022, no. 1, p. 1028175, 2022, doi: https://doi.org/10.1155/2022/1028175.

[15] W. Long *et al.*, "Unified Spatial-Temporal Neighbor Attention Network for Dynamic Traffic Prediction," *IEEE Trans Veh Technol*, vol. 72, no. 2, pp. 1515–1529, 2023, doi: 10.1109/TVT.2022.3209242.

[16] Ma. J. Hemández-Molinos, Á. J. Sánchez-García, and R. E. Barrientos-Martínez, "Classification Algorithms for Software Defect Prediction: A Systematic Literature Review," in *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2021, pp. 189–196. doi: 10.1109/CONISOFT52520.2021.00034.

[17] R. Aflaha, R. Herteno, M. R. Faisal, F. Abadi, and S. Saputro, "Effect of SMOTE Variants on Software Defect Prediction Classification Based on Boosting Algorithm," *Jurnal Ilmiah Teknik Elektro Komputer dan Informatika*, vol. 10, pp. 201–216, Mar. 2024, doi: 10.26555/jiteki.v10i2.28521.

[18] A. Balogun *et al.*, "SMOTE-Based Homogeneous Ensemble Methods for Software Defect Prediction," 2020, pp. 615–631. doi: 10.1007/978-3-030-58817-5_45.

[19] V. B. Singh, S. Misra, and M. Sharma, "Bug Severity Assessment in Cross Project Context and Identifying Training Candidates," *Journal of Information & Knowledge Management*, vol. 16, no. 01, p. 1750005, 2017, doi: 10.1142/S0219649217500058.

[20] A. N. Rao Moparthi and B. Dr. N. Geethanjali, "Design and implementation of hybrid phase based ensemble technique for defect discovery using SDLC software metrics," in *2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, 2016, pp. 268–274. doi: 10.1109/AEEICB.2016.7538287.

[21] F. Matloob *et al.*, "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 98754–98771, 2021, doi: 10.1109/ACCESS.2021.3095559.

[22] N. Grattan, D. Alencar da Costa, and N. Stanger, "The need for more informative defect prediction: A systematic literature review," *Inf Softw Technol*, vol. 171, p. 107456, 2024, doi: https://doi.org/10.1016/j.infsof.2024.107456.

[23] A. Hardoni, "Integrasi SMOTE pada Naive Bayes dan Logistic Regression Berbasis Particle Swarm Optimization untuk Prediksi Cacat Perangkat Lunak," *Jurnal Sistem dan Teknologi Informasi (Justin)*, vol. 9, p. 144, Feb. 2021, doi: 10.26418/justin.v9i2.43173.

[24] K. Suryadi, R. Herteno, S. W. Saputro, M. R. Faisal, and R. Nugroho, "Comparative Study of Various Hyperparameter Tuning on Random Forest Classification With SMOTE and Feature Selection Using Genetic Algorithm in Software Defect Prediction," *Journal of Electronics Electromedical Engineering and Medical Informatics*, vol. 6, pp. 137–147, Mar. 2024, doi: 10.35882/jeeemi.v6i2.375.

[25] Y. Zhao, Z. Huang, L. Gong, Y. Zhu, Q. Yu, and Y. Gao, "Evaluating the Impact of Data Transformation Techniques on the Performance and Interpretability of Software Defect Prediction Models," *IET Software*, vol. 2023, no. 1, p. 6293074, 2023, doi: https://doi.org/10.1049/2023/6293074.

[26] L. B. V de Amorim, G. D. C. Cavalcanti, and R. M. O. Cruz, "The choice of scaling technique matters for classification performance," *Appl Soft Comput*, vol. 133, p. 109924, 2023, doi: https://doi.org/10.1016/j.asoc.2022.109924.

[27] S. Susan and A. Kumar, "The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art," *Engineering Reports*, vol. 3, no. 4, p. e12298, 2021, doi: https://doi.org/10.1002/eng2.12298.

[28] N. Pudjihartono, T. Fadason, A. Kempa-Liehr, and J. O'Sullivan, "A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction," *Frontiers in Bioinformatics*, vol. 2, p. 927312, Feb. 2022, doi: 10.3389/fbinf.2022.927312.

[29] Y. Feng, L. Feng, S. Liu, S. Kwong, and K. C. Tan, "Towards multi-objective high-dimensional feature selection via evolutionary multitasking," *Swarm Evol Comput*, vol. 89, p. 101618, 2024, doi: https://doi.org/10.1016/j.swevo.2024.101618.

[30] J. Xie, X. Li, L. Gao, and L. Gui, "A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems," *J Manuf Syst*, vol. 71, pp. 82–94, 2023, doi: https://doi.org/10.1016/j.jmsy.2023.09.002.

[31] H. Alabool, D. Al- Arabiat, L. Abualigah, and A. A. Heidari, "Harris hawks optimization: a comprehensive review of recent variants and

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

applications," *Neural Comput Appl*, vol. 33, Feb. 2021, doi: 10.1007/s00521-021-05720-5.
[32] M. H. Nadimi-Shahraki, H. Zamani, Z. Asghari Varzaneh, and S. Mirjalili, "A Systematic Review of the Whale Optimization Algorithm: Theoretical Foundation, Improvements, and Hybridizations," *Archives of Computational Methods in Engineering*, vol. 30, no. 7, pp. 4113–4159, 2023, doi: 10.1007/s11831-023-09928-7.
[33] M. Al Shinwan *et al.*, "Dragonfly algorithm: a comprehensive survey of its results, variants, and applications," *Multimed Tools Appl*, vol. 80, Apr. 2021, doi: 10.1007/s11042-020-10255-3.
[34] P. Qin, H. Hu, and Z. Yang, "The improved grasshopper optimization algorithm and its applications," *Sci Rep*, vol. 11, p. 23733, Apr. 2021, doi: 10.1038/s41598-021-03049-6.
[35] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing Software Defect Prediction Models: Integrating Hybrid Grey Wolf and Particle Swarm Optimization for Enhanced Feature Selection with Popular Gradient Boosting Algorithm," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:269701785
[36] J. Jui, M. M. I. Molla, M. A. Ahmad, and I. Hettiarachchi, "Recent Advances and Applications of the Multi-verse Optimiser Algorithm: A Survey from 2020 to 2024," *Archives of Computational Methods in Engineering*, Apr. 2025, doi: 10.1007/s11831-025-10277-w.
[37] D. Rawat and P. Singh, "An Effective Algorithm using Moth Flame Optimization (MFO) for Numerical Expression Solutions.," *International Journal For Multidisciplinary Research*, 2024, [Online]. Available: https://api.semanticscholar.org/CorpusID:268357608
[38] T. Thaher, A. A. Heidari, M. Mafarja, and J. Dong, "Binary Harris Hawks Optimizer for High-Dimensional, Low Sample Size Feature Selection," 2020, pp. 251–272. doi: 10.1007/978-981-32-9990-0_12.
[39] T. Thaher and N. Arman, "Efficient Multi-Swarm Binary Harris Hawks Optimization as a Feature Selection Approach for Software Fault Prediction," Feb. 2020. doi: 10.1109/ICICS49469.2020.239557.
[40] X. Xiaolong, C. Wen, and W. Xinheng, "RFC: A feature selection algorithm for software defect prediction," *Journal of Systems Engineering and Electronics*, vol. 32, no. 2, pp. 389–398, 2021, doi: 10.23919/JSEE.2021.000032.
[41] H. Ghinaya, R. Herteno, M. R. Faisal, A. Farmadi, and F. Indriani, "Analysis of Important Features in Software Defect Prediction Using Synthetic Minority Oversampling Techniques (SMOTE), Recursive Feature Elimination (RFE) and Random Forest," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 3, pp. 276–288, May 2024, doi: 10.35882/jeeemi.v6i3.453.
[42] J. Magidi, L. Nhamo, S. Mpandeli, and T. Mabhaudhi, "Application of the Random Forest Classifier to Map Irrigated Areas Using Google Earth Engine," *Remote Sens (Basel)*, vol. 13, no. 5, 2021, doi: 10.3390/rs13050876.
[43] A. Przybyś-Małaczek, I. Antoniuk, K. Szymanowski, M. Kruk, and J. Kurek, "Application of Machine Learning Algorithms for Tool Condition Monitoring in Milling Chipboard Process," *Sensors*, vol. 23, p. 5850, Apr. 2023, doi: 10.3390/s23135850.
[44] M. Azzeh, Y. Elsheikh, A. Nassif, and L. Angelis, "Examining the Performance of Kernel Methods for Software Defect Prediction Based on Support Vector Machine," *Sci Comput Program*, vol. 226, p. 102916, Feb. 2022, doi: 10.1016/j.scico.2022.102916.
[45] A. Alazba and H. Aljamaan, "Software Defect Prediction Using Stacking Generalization of Optimized Tree-Based Ensembles," *Applied Sciences*, vol. 12, no. 9, 2022, doi: 10.3390/app12094577.
[46] M. Altalhan, A. Algarni, and T. Monia, "Imbalanced Data Problem in Machine Learning: A Review," *IEEE Access*, vol. PP, p. 1, Apr. 2025, doi: 10.1109/ACCESS.2025.3531662.
[47] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Yasin Ghadi, and M. Amir Khan, "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning," *PeerJ Comput Sci*, vol. 10, p. e1860, Feb. 2024.
[48] A. O. Balogun *et al.*, "Software Defect Prediction Using Wrapper Feature Selection Based on Dynamic Re-Ranking Strategy," *Symmetry (Basel)*, vol. 13, no. 11, 2021, doi: 10.3390/sym13112166.

AUTHOR BIOGRAPHY

Achmad Fauzan Luthfi is a Computer Science student at the Faculty of Mathematics and Natural Sciences, Universitas Lambung Mangkurat, where he has been pursuing his studies since 2021. He has a strong interest in software engineering and software defect prediction (SDP). In 2023, he participated in the Bangkit program (batch 2), organized by Google Indonesia, specializing in Machine Learning with TensorFlow. In mid-2024, he completed a

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

four-month internship, during which he was involved in an Android application development project using Flutter. He is currently exploring backend development, focusing on JavaScript and its various supporting frameworks. He can be contacted via email at: 2111016210004@mhs.ulm.ac.id.

Rudy Herteno received his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011. After completing his studies, he worked as a software developer for several years to gain more experience in the field. During this period, he developed various software applications, particularly to support the needs of local governments. In 2017, he obtained a master's degree in Informatics from STMIK Amikom University. Currently, he is a lecturer in the Computer Science program at Lambung Mangkurat University. His research interests include software engineering, software defect prediction, and deep learning, aiming to improve software quality, optimize error detection in systems, and develop artificial intelligence-based solutions.He can be contacted at email: rudy.herteno@ulm.ac.id.

Friska Abadi finished his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011. Subsequently, in 2016, he obtained his master's degree from the Department of Informatics at STMIK Amikom, Yogyakarta. Following that, he joined Lambung Mangkurat University as a lecturer in Computer Science. As a lecturer he teaches programming. Apart from that, he also carries out research and community service. Other activities as an application developer, whether using a web or mobile platform. Currently, he holds the position of head of the software engineering laboratory. His current area of research revolves around software engineering and also interested in machine learning. He can be contacted at email: friska.abadi@ulm.ac.id.

Radityo Adi Nugroho received his bachelor's degree in Informatics from the Islamic University of Indonesia and a master's degree in Computer Science from Gadjah Mada University. Currently, he is an assistant professor in the Department of Computer Science at Lambung Mangkurat University. His research interests include software defect prediction and computer vision. He can be contacted at email: radityo.adi@ulm.ac.id.

Muhammad Itqan Mazdadi, a lecturer in the Department of Computer Science, Lambung Mangkurat University. His research interest is centered on Data Science and Computer Networking. Before becoming a lecturer, he completed his undergraduate program in the Computer Science Departmentat Lambung Mangkurat University In 2013. He then completed his master's degree from Department of Informatics at Islamic Indonesia University, Yogyakarta. Currently, he serves as the Secretary of the Computer Science Department at Lambung Mangkurat University. He can be contacted at email: mazdadi@ulm.ac.id.

Professor Dr. Vijay Anant Athavale is a distinguished academic and professional with extensive experience in computer science and engineering. He holds a Ph.D. in Computer Science from Barkatullah University, Bhopal, and has served in various prestigious roles, including Dean of Engineering and Professor at Panipat Institute of Engineering & Technology, Haryana and Department of Computer Science and Engineering, Walchand Institute of Technology, Solapur, Maharashtra, India. Dr. Athavale has been a key figure in numerous institutions, contributing significantly to their academic and administrative advancements. He is a life member of several professional bodies, such as the Computer Society of India and ISTE. His research interests include machine learning, IoT, and data management, with numerous publications and patents to his name. Dr. Athavale has also chaired and organized several international conferences, reflecting his commitment to advancing technology and education.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru, Indonesia.
https://doi.org/10.35882/f2140043