International Journal of Computational Intelligence Systems (2025) 18:14
https://doi.org/10.1007/s44196-024-00716-0

RESEARCH ARTICLE

Adaptive Ensemble Learning Model-Based Binary White Shark Optimizer for Software Defect Classification

Jameel Saraireh[1] · **Mary Agoyi**[2] · **Sofian Kassaymeh**[3,4]

Received: 8 July 2024 / Accepted: 1 December 2024

Abstract
Software dominates modern enterprises, affecting numerous functions. Software firms constantly experiment with new methodologies to define and assess software quality to stay competitive and ensure excellence. Software engineering uses fundamentals and cutting-edge technology to develop great software. In recent decades, Data-mining techniques and machine learning for classifying problematic software projects have emerged to improve software quality. ML approaches, especially ensemble learning models, are becoming fundamental to software engineers' daily jobs. This work created a binary white shark optimizer (WSO) to optimize standard ensemble learning models. The objective is to identify the most suitable ensemble number for weak learners to maximize accuracy on benchmark datasets. The EM model uses 14 weak learners. Twenty-one experimental runs are performed on 15 software-defective module datasets. The optimized ensemble model outperforms the standard Ensemble learning model in AUC-ROC, Accuracy, Precision, Recall, F1-Score, and Specificity. The enhanced model has an average accuracy of 86%, compared to 76% for the standard ensemble model across all datasets. The optimized model outperformed the conventional ensemble for the same datasets, with an average AUC of 72% compared to 61% for the standard ensemble. The optimized model was more stable than the standard model, with an STD of 5.53E−03 vs 7.24E−02 for the ensemble model. The WSO optimization process strengthens and generalizes optimizeels. The study suggests that evolutionary metaheuristic approaches can enhance EM models' accuracy, trustworthiness, and adaptability.

Keywords Binary white shark optimizer · Ensemble learning · Software defect classification · Software quality

1 Introduction

Software is essential to the success of all today's organizations, adding value to all business areas. All of the company's business processes rely heavily on software. Software development companies always seek new methods and techniques to gain an edge. According to expert [1], software is essential for the operation of nearly all functions of an organization. Many software systems assist people, businesses, and communities in taking advantage of opportunities or solving problems. For the UK's Office of Government Commerce (OGC), a benefit is a demonstrable improvement from an outcome perceived by one or more parties that support the organization's goals [2]. Software failures can cost a business a lot of money. An organization must also design customer-satisfying software. Since humans build software, it can have errors during the development life cycle. Diverse techniques and strategies have surfaced to ensure dependable and superior software development [3].

Mary Agoyi and Sofian Kassaymeh have contributed equally to this work.

✉ Jameel Saraireh
jameel.work85@gmail.com; 21908602@student.ciu.edu.tr

Mary Agoyi
magoyi@ciu.edu.tr

Sofian Kassaymeh
s.kassaymeh@aut.edu.jo

1 Department of Management Information Systems, Cyprus International University, Haspolat, Lefkoşa, Nicosia, North Cyprus, Turkey

2 Department of Information Technology, Cyprus International University, Haspolat, Lefkoşa, Nicosia, North Cyprus, Turkey

3 Jadara University Research Center, Jadara University, Irbid, Jordan

4 Software Engineering Department, Aqaba University of Technology, South Cost, Aqaba, Jordan

Published online: 23 January 2025

Various models and standards exist for defining and assessing software quality. The latest are ISO/IEC 25010 and ISO 5055 [4]. Although quality is determined by eight quality criteria in ISO/IEC 25010, for instance, maintainability Quality is defined by ISO/IEC 5055 as "weaknesses" that imperil software integrity, trustworthiness, effectiveness, and ability to be maintained [5]. Software quality has internal and external properties. Software artifacts assess internal quality criteria like size, coupling, and cohesion. Environmental information is needed to measure external quality attributes, including maintainability, reusability, and reliability [6]. Software quality is a priority for the industry. Models like ISO/IEC 25,010 (ISO, IEC25010:2011 2011) [5] consider software reliability as a quality factor. Per the IEEE software reliability standard (IEEE Reliability Society 2016) [7]: (1) The probability of software operating without error in a specific environment for an agreed-upon period is known as software reliability. (2) Modeling software reliability outlines the software failure process based on factors, including defect introduction, removal, and operating environment [8]. Software developers aim to create reliable, low-cost, high-quality software. Software reliability is the chance of a software operation being completed without failure within a particular time frame in a given environment. Software development prioritizes reliability to ensure that the software works as intended [9].

Software developers make decisions based on software reliability. Software reliability research generally assumes flawless debugging, where defects can be promptly eliminated or fixed [10]. A piece of software is considered "dependable" if it can function continuously for an exact amount of time [11]. Evaluation of software reliability quantitatively is crucial during the SDLC testing phase to reduce failure risk, optimize resource utilization, manage costs, and estimate faults, resulting in a highly reliable software system [12]. Testing environment, strategy, and resource allocation might affect release time. Software release decisions become increasingly complicated and vital; when a software creator makes an intentional decision, the client or end-user faces significant financial losses. Software release decisions involve balancing a premature release, which will benefit from a faster market launch, with product release postponement to maintain reliability. Software developers must pay for bug fixes if a product is published too soon. For optimal software release timing, reliability and cost must be considered [13]. By recognizing troublesome modules in their early stages, developers can better deploy Assets for quality assurance and enhance software quality during development and maintenance [14]. To understand software quality thoroughly, we must analyze it from several viewpoints, such as software engineering, statistics, and machine learning.

Software quality improvement methods include Capability Maturity Model Integration (CMMI), Total Quality Management, Six Sigma, and Lean Production. Software businesses have started Six Sigma programs to optimize procedures used in software development throughout life cycle phases to provide high-quality products. Evaluating software project development costs and effort is another crucial task in software project management. Organizations lose contracts or fail at software project management due to underestimating and overestimating. Software project managers struggle with cost/effort estimation, which is crucial to project success [15]. To boost software quality, software effort estimation takes into account the number of errors discovered and fixed during the software development life cycle [16–18]. Software defects are typically deviations from specifications or requirements [19]. Such faults may cause failures or unexpected effects. Software quality assurance tasks like defect prediction, code review, and unit testing reduce failures and increase software quality. Such efforts often consume 80% of a project's budget. Software engineers prioritize inspecting modules with the most flaws to reduce costs [20].

Cross Project Defect Prediction (CPDP) is a technique for constructing a defect prediction model based on source projects and subsequently implementing it in a target project. Selecting an appropriate training project for developing a predictor model in CPDP is challenging due to the emergence of new software projects [21]. SDP studies can be categorized according to their contexts and the diverse alternatives in the procedural processes. Literature has conventionally utilized two fundamental situations for Software Defect Prediction (SDP): Defect prediction within and across projects (WPDP and CPDP). WPDP uses a project's history data, specifically various versions, to forecast the problematic components [22]. WPDP specifically focuses on fault predictions within the identical software project on which it is trained [23]. Consequently, both the training and testing datasets pertain to the same project. Conversely, CPDP utilizes data from source projects to train an SDP model, which is subsequently employed to predict the problematic aspects of a target project. This transfer learning-based method is highly beneficial for projects with constrained labeled data. This technique's primary challenge is reducing the disparity in feature distribution between source and target projects [22, 24]. This study employs CPDP as the writers trained and identified necessary metrics across 15 projects.

Predicting defect-prone areas of software before their identification through substantial effort is challenging. The primary challenge of Software Defect forecasting is the identification of defective segments within the source code while enhancing the accuracy of fault prediction. A multitude of strategies and techniques have been suggested and documented in the literature to achieve this purpose. Numerous studies employ learning-based techniques to enhance SDP accuracies; nevertheless, The conceptual description of source code is the focus of some research. To cre-

ate efficient SDP models, researchers employ ML and DL approaches [22]. Software defect prediction constitutes a highly dynamic domain within software engineering. Recently, several researchers introduced Just-in-Time Defect Prediction Technology. The timely defect prediction system has gained recognition for its accuracy and simplicity. This method can ascertain the presence of a software defect in any code update supplied by a developer. Moreover, the process is rapid and easy to monitor. The primary challenge is that the category imbalance within the data set affects the predictive accuracy of Just-in-Time software. Generally, 20% of software engineering issues arise in 80% of modules, with code modifications that do not produce a substantial portion of faults. Consequently, there exists a disparity in the data set, especially when there is a large gap between two or more classes. This affects the model's ability to accurately anticipate the classes to use for classification [25]. A crucial element in the prompt delivery of superior software products is software defect prediction (SDP). It anticipates error-prone modules at the first phases of development, which could result in significant damage or potential program failure subsequently. Consequently, it reduces the total software development expenses while facilitating the focused testing of these troublesome modules, ensuring the superior quality of the final product. Support vector machines (SVMs) are extensively employed in SDP. The dataset's imbalanced distribution of defective and non-defective modules impairs the accuracy of support vector machines (SVMs). This research introduces an innovative filtering method (FILTER) for effective SVM-based defect prediction. Employing the recommended filtering method, classifiers based on support vector machines (SVM)—linear, polynomial, and radial basis function—are constructed, and their performances are evaluated across five datasets. There is a 16.73% improvement in accuracy, a 16.80% improvement in AUC, and a 7.55% improvement in the F-measure of the SVM-based SDP model due to the proposed FILTER [26]. Software defect prediction (SDP) is an innovative technological approach to forecasting software defects within the software development life cycle. Many studies on SDP have been completed previously; however, their results differ across various datasets, making them unreliable for SDP in unspecified software projects. In contrast to a singular classifier, the hybrid methodology that integrates the selection of features through machine learning for SDP can yield superior efficacy by utilizing many strategies to enhance prediction accuracy for a specific dataset. Individual machine learning-based models for software defect prediction encounter three primary issues: elevated dimensionality of feature parameters, protracted detection durations, and vulnerabilities within software projects. This paper proposes a hybrid model utilizing an Extreme Gradient Boost (XGB) classifier with enabled feature selection to address these concerns. The proposed model was executed utilizing the refined NASA MDP datasets, and its efficacy was evaluated through various performance metrics, including F-score, accuracy, and MCC. The performance of the proposed model surpasses that of state-of-the-art strategies lacking feature selection. The proposed model outperformed all other predictive methods, as indicated by the results [27]. In software engineering, forecasting software defects is essential for enhancing the quality of software systems, which is a critical and expensive component of the software development lifecycle. As we increasingly employ software systems, their dependencies and complexities escalate, fostering an environment prone to breakdowns. Software defects lead to erroneous outcomes and actions. More crucial than defects is the ability to identify them before their emergence. Consequently, identifying (and forecasting) software issues enables software managers to deploy resources more effectively during the maintenance and testing phases. The literature presents numerous proposals for forecasting software defects. The authors of this research performed a comparative examination of machine learning-based software defect prediction systems utilizing the public datasets PC1, JM1, KC1, KC2, and CM1 from the PROMISE repository. The researchers evaluated ten learning algorithms: Naive Bayes, K-Nearest Neighbor, Decision Tree, Support Vector Machine, Extra Trees, Random Forest, Multi-Layer Perceptron, Adaboost, Gradient Boosting, and Bagging. The experimental results demonstrated that the proposed models provide adequate accuracy levels for software defect prediction, hence enhancing product quality [28]. Artificial intelligence has invaded several sectors. Many manufacturing robots work instead of humans. These robots have ML-trained brains. AI and software engineering (SE) are becoming more integrated, but the gap is still substantial compared to AI and additional sciences. Machine learning algorithms have become increasingly important in the software industry and software professionals' lives [29]. Machine learning and data mining are concerned with software engineering techniques, particularly SDP. SDP has been classified using Decision Tree, Support Vector Machine, and Logistic Regression [30–33]. Many software defect prediction techniques have been presented to boost software quality in recent decades. Machine learning is becoming more prevalent. Supervised methods require labels for training data, whether flawed or not, while unsupervised methods do not [34]. Software defects, such as faults or failures in computer systems or applications, are prevalent and lead them to act unexpectedly, which lowers software quality [35].

The classification of software defects helps project managers predict vulnerabilities before product introduction, improving validation and testing. Finding code errors early in software development may improve both reliability and performance. These methods help developers discover the most likely defects, improving program performance. Defec-

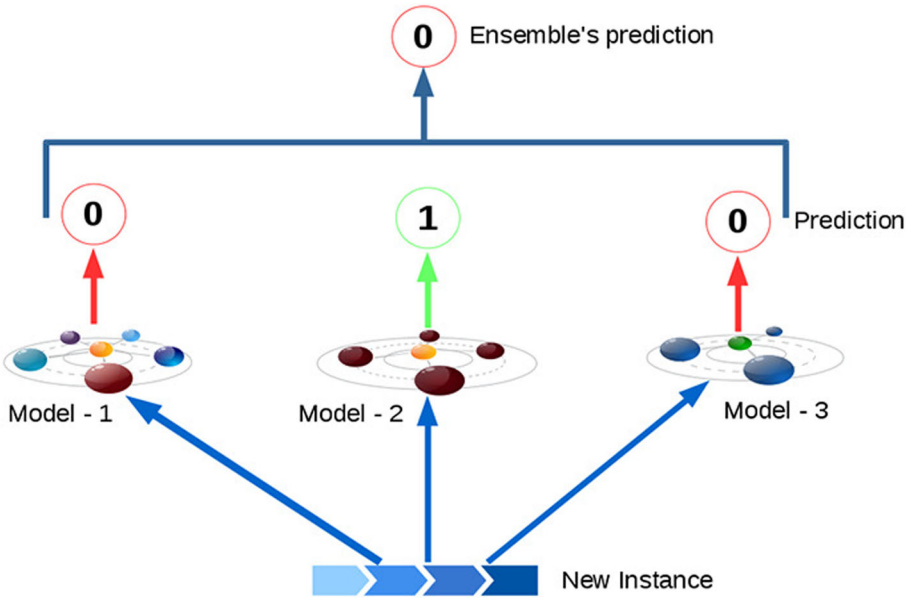

Fig. 1 Ensemble majority voting model

tive parts have been classified and predicted using many ML techniques [36]. Early defect detection improves software quality at minimal expense. Pemmada et al. [37] endeavor to create a neural network model based on correlation with the intent of spotting software defects.

Machine learning has several methods to classify and predict defect software; one of these methods is ensemble learning. Ensemble Learning has demonstrated efficacy and practicality in various problem domains and important Practical uses [38].

Ensemble learning algorithms have demonstrated compelling performance and have attracted interest in predicting software errors [39]. Ensemble learning is a method that entails creating many classifiers or a set of base learners and combining their outputs to reduce overall variation. Combining many classifiers or base learners significantly enhances the accuracy of findings compared to employing just one at a time. Research has demonstrated that ensemble approaches can improve the predictive accuracy of machine learning models across many tasks, such as classification, regression, and outlier detection [40]. Ensemble learning consists of different methods, e.g., (1) Bagging has the benefit of lowering variance, eliminating the possibility of overfitting due to the advantageous solution of the Bagging method used in this research study. In addition to this, it is effective when applied to high-dimensional data [41]. Random Forests (RF) algorithm [42] is an excellent example of bagging. (2) Boosting makes it easier to analyze the model and contributes to reducing variance and bias in an ensemble of machine learning models. (3) Stacking provides a more in-depth understanding of the data, making it more accurate and efficient [43]. (4) A voting classifier employs two techniques: hard voting and soft voting. Hard voting involves making the final forecast based on a majority vote, where the class prediction is picked by the aggregators that the base models most frequently select. At the same time as it yields satisfactory outcomes, this method has the benefit of having a computational cost that is almost negligible [44]. This research adopted this technique. Base models in soft voting must possess the Predictive probability technique. The voting classifier outperforms conventional basic models by aggregating predictions from several models [45]. Figure 1 shows the process of majority ensemble voting.

The criteria for selecting the appropriate ensemble approach often involve the problem being addressed, the dataset's characteristics, and the available computer resources. Boosting has emerged as the most often-used ensemble learning technique. It constructs a robust classifier by successively adapting and integrating several similar weak learners. Sequential learning enhances the ability to approximate and generalize, as demonstrated by [46]. Boosting provides superior accuracy, performance, flexibility, and interpretability compared to ensemble learning approaches such as bagging and stacking [47]. Software Fault Prediction (SFP) is a crucial method for early identification of defective software components, including erroneous classes or modules, at various points throughout the SDLC. In [48], the authors suggest a machine learning structure to accommodate SFP. Pre-processing and resampling procedures are first used to get the SFP datasets ready for use with ML algorithms. The subsequent seven classifiers are evaluated: Support Vector Machine (SVM), Naive Bayes (NB), Linear Discriminant Analysis (LDA), Linear Regression (LR), Decision Tree (DT), K-Nearest Neighbors (KNN), and Random Forest (RF). The RF classifier outperforms all other classifiers in eliminating superfluous or redundant informa-

tion. The binary whale optimization algorithm increased RF performance and decreased dimensionality by removing excessive data. For better BWOA, try using the Grey Wolf Optimizer (GWO) or the Harris Hawk Optimization (HHO) exploration processes. A method called SBEWOA is offered. Software project datasets spanning various sizes and complexity are available in the PROMISE repository. The suggested SBEWOA beat nine other feature selection methods on all datasets tested. Accuracy, feature number, and fitness function are the metrics used to evaluate the algorithms. The two-tailed P values of the Wilcoxon signed-rank statistical test support this. Conclusion: The suggested method is an efficient alternative machine learning approach for software fault prediction applicable to similar difficulties in software engineering. Due to their consistent and robust performance, machine learning (ML) techniques were employed to tackle the software failure prediction (SFP) issue. Multilayer perceptron (MLP) neural networks rank among the most potent machine learning models for predictive tasks. Regrettably, MLP has persistent deficiencies attributable to the gradient-descent learning mechanism, which is prone to become trapped in local minima, leading to erroneous control parameters. Al-Laham et al. [49] introduce an enhanced version of the Salp Swarm Optimizer (SSA), a metaheuristic swarm intelligence method, to improve MLP for solving SFP. The MLP learning methodology has been enhanced with SSA to resolve these issues. The principal benefit of this algorithm is its capacity to circumvent local minima through its convergence behavior. Two alterations were implemented in the SSA optimization loop to integrate SSA functionalities with MLP. The initial enhancement is elitism (SSA-elitism), followed by the second enhancement, MSSA, or search optimization. The efficacy of the proposed SSA versions is evaluated against 18 benchmark SFP datasets utilizing ROC, sensitivity, specificity, and accuracy metrics. Numerous assessments and verifications were performed by juxtaposing the results of the generated versions with those of the traditional MLP, SSA, and ten cutting-edge methodologies. The assessments and validation results indicate that the suggested models can effectively tune MLP parameters, improving predictive quality. The adaptive variable sparrow search algorithm enhanced the global optimization of the original sparrow search method by employing adaptive hyperparameters and variable logarithmic spirals. AVSSA's assessment of the eight benchmark functions produced outstanding outcomes. The efficacy of the typical defect detection technique is contingent upon asymmetry in data distribution. The study improves ensemble learning as a predictor for Bagging ensemble learning (AVSEB) by adaptive variable sparrow search. In [50], a novel method for anticipating software defects via ensemble learning is presented. The model first employed the unstable cut-points technique to preprocess the Bagging sample set. The adaptive variable sparrow search approach is then utilized to optimize the ensemble learning. The voting process is used to obtain predictive outcomes for software defects. The experimental results indicate that our proposed algorithm's evaluation metric surpasses four other sophisticated comparative algorithms across 15 software defect datasets. The algorithm suggested in this study surpasses previous advanced prediction algorithms regarding statistical significance, as evidenced by the results of Friedman's ranking and Holm's post hoc test. Software reliability constitutes a critical dimension of software quality. A testing phase with discovered and corrected problems is incorporated into software development to enhance reliability. The fault detection process (FDP) constitutes a component of the fault correction process (FCP) utilized to develop the software reliability growth model. This integration is challenging due to various factors, including staff shortages and dependability on faults. It constrains the utility of the analytical model. The application of data-driven methodologies, including Artificial Intelligence (AI) technology, obviates the necessity for precise FCP and FDP assumptions. The researchers proposed a hybrid long short-term memory (LSTM) model utilizing the BrainStorm Optimization and Late Acceptance Hill Climbing (BSO-LAHC) method for a stepwise prediction approach in software problem discovery and rectification. The technique for identifying and rectifying defects significantly influences the assessment of testing efforts. Compared to current methodologies, the proposed hybrid utilizing the BSO-LAHC algorithm yielded superior outcomes when implemented with Firefox and the problem-tracking system Bugzilla. The efficacy of the proposed paradigm is substantiated by empirical study. The mean square error performance for the Bugzilla and Firefox datasets is 1.92 and 21.44, respectively. Moreover, the suggested technique is more cost-effective and necessitates reduced execution time. In Bugzilla 5.0.4, releases 2 and 3 exhibited determination coefficients of 99.2% and 98.9%, respectively. The FCP is 27% more effective than previous approaches, whereas the FDP is 32% more effective [51]. Feature selection is a vital and challenging phase in classification technology. It is employed to diminish the complexity of a dataset and remove extra elements. The authors utilized KNN, Naive Bayes, and Decision Tree classifiers. The authors developed a fitness function and employed a two-step pheromone update approach to remove redundant characteristics efficiently. This program emulates real ants, which seek the most efficient route to a food source by utilizing pheromone concentration. The authors analyzed 12 distinct datasets and juxtaposed them with fitness graphs. Each graph illustrates the efficacy of ant colony optimization in conjunction with diverse classifiers. The authors have constructed a table illustrating the predictive accuracy of several classifiers utilizing the method [52]. As a result, we con-

clude that maintaining software quality can be accomplished through various techniques, the ensemble method being the most significant of these alternatives. This research project aims to demonstrate a sophisticated ensemble-based software defect prediction model that intelligently incorporates many classifiers. The proposed methodology consists of three essential stages that efficiently detect faulty software. During the initial stage, a comprehensive collection of 14 ML classifiers is utilized, which includes various techniques, such as KNeighbors Classifier, Naive Bayes, Decision Tree Classifier, Gradient Boosting Classifier, Random Forest Classifier, AdaBoost Classifier, ExtraTrees Classifier, Ridge Classifier, Logistic Regression, MLP Classifier, Quadratic Discriminant Analysis, Bagging Classifier, HistGradient Boosting Classifier, Support Vector Machine, and others. The classifiers, which are weak learners, are subjected to tweaking through a bagging-ensemble methodology to improve accuracy. Following this, the predicted precision of the individual classifiers is combined using a voting ensemble technique to obtain the ultimate predictions. During the third step, the binary-WSO optimizer is used repeatedly to improve the model's accuracy and decrease the computational time it takes to work. This is achieved by establishing the most effective number of weak learners (classifiers) for each unique problem studied. This optimized ensemble technique dramatically enhances the accuracy and dependability of fault forecasts. Fifteen historical benchmark defect datasets obtained from respected archives, such as NASA MDP, Relink, and Softlab, are employed to validate the proposed defect prediction system. Combined with the binary-WSO optimizer, the intelligent system exhibits remarkable accuracy and execution efficiency across all datasets. This surpasses the performance of basic classifiers when employing ensemble techniques.

1.1 Research Questions

The purpose of this study is to provide answers to the following questions:

- **RQ1:** What impact does the binary-WSO optimizer have on determining the ideal number of weak learners in ensemble models to improve accuracy and processing efficiency (Execution time)?
- **RQ2:** How does the effectiveness of the binary-WSO-based ensemble learning model compared to the state-of-the-art techniques currently in use for software fault prediction?
- **RQ3:** How far can the results of the binary-WSO-optimized ensemble learning model be applied to software-defective modules from different repositories?

1.2 Contribution

The following are the research's principal contributions:

- Tackling the software defect prediction problem by proposing the ensemble learning model (EM).
- Ensuring high prediction accuracy by utilizing most classifiers available in the literature.
- Developing a binary version of the White Shark Optimizer (WSO).
- Employing the binary-WSO as an optimizer for the developed EM model to identify the optimal number of weak learners (classifier).
- Boosting the EM model's prediction accuracy and computational efficiency using the binary-WSO optimizer.
- Generalizing obtained results over 15 software-defective module datasets from different repositories.
- Comparing the proposed method with **14** state-of-the-art algorithms.

In this work, we examine a novel strategy: how to utilize Binary White Shark Optimizer to enhance the performance of an ensemble model utilizing 15 different datasets in software defect classification. Specifically, we focus on how to increase the performance of ensemble learning models. When contrasted with metaheuristic algorithms used in the past, this work is anticipated to present an exceptional way. Although the dataset here has imbalance problems, this problem will be fixed throughout the project's preparation phase. As authors, we are excited about doing a novel study that can be generalized for future investigations, mainly when BWSO is utilized.

1.3 Article Organization

The remainder of the article is divided into the following sections: The research background of Ensemble Methods and white shark optimizer is detailed in Sect. 2. A literature review for proposed studies on the investigated problem is presented in Sect. 3. The methodology and proposed model are provided in Sect. 4. Modeling and findings of the experimental results are discussed in Sect. 5. The research concludes in Sect. 6.

2 Background

Due to the cognitive nature of the method of generating software and the growing complexity of software products, software errors are becoming more common, leading to low-quality software requiring around 2.8 trillion USD to correct. The discipline of prediction of software defects (SDP) was created to improve software quality. Its creative endeavor

to separate problematic software units allows for eliminating defects and more efficient use of resources for software development and maintenance tasks [53].

2.1 Software Defected Prediction

Software is essential to creating a global village. Almost all businesses, international corporations, the e-commerce sector, social networking, general-purpose, personal use, software for medical devices, and other organizations rely on software platforms growing exponentially over time [54]. The software defect process is shown in Fig. 2.

A software defect is an error, fault, or flaw in a program with unfavorable consequences. Errors in programming are known as software defects, and they might manifest themselves as a result of deficiencies in the software's requirements, design, or source code. The presence of defects has a detrimental impact on the quality and reliability of software. As a result, they lead to a rise in maintenance expenses and the effort required to resolve problems [36]. By predicting errors that may occur inside a software project, software defect prediction (SDP) allows software engineers to optimize resource allocation to improve software quality [55]. Nevertheless, managers must develop a prediction model to help identify defective modules, because it takes time to anticipate defect density before testing them. This procedure can lower testing costs and increase the use of testing resources [56]. To predict which modules are more likely to have mistakes and defects before the testing phase begins, the software development process is significantly dependent on software defect prediction (SDP). Focusing on testing activities on the modules that are expected to be defective will lower the cost of constructing software. Nonetheless, it ensures immediate delivery of a superior finished product [57].

Over the past twenty years, the problem of software defect prediction has garnered significant attention from scholars and grown in significance. Software projects can be classified as non-defective or defective (binary class classification), their number of flaws can be predicted, or their severity can be predicted, using a software defect prediction model [58]. Over the past three decades, software engineering research has received much attention in determining which software elements, like files, classes, and processes, are likely to be defect-prone. It is beneficial to consistently discern between components prone to defects and those that are clean, as this allows quality assurance resources to be spent more efficiently [59]. Software defect prediction is a crucial component in raising the caliber of software products. It speeds up the development process and lowers development costs. It provides the ability to evaluate a module's susceptibility to errors and predict which software component will need additional testing and quality assurance (QA) resources. It makes it possible to prevent software failure in the future by implementing proactive measures during the development stage [57].

According to [50, 57, 60, 61], software defect prediction (SDP) is one area of software engineering that is now the subject of a great deal of research. The main application of the SDP model is to forecast which modules are prone to defects. The SDP model is usually constructed using classification algorithms that rely on past data. To control software fault rates, software defect prediction technology may use software module measurement data during software development to proactively find problematic modules and allocate test resources efficiently.

Reports on bugs are employed to record issues that are found during software development and upkeep. A problem with the potential to cause significant harm is described in a high-impact bug report (HBR), which appears after delivery. HBRs must be identified early from the bug pool to ensure functionality [62].

Advancements in machine learning (ML) facilitate a shift from the conventional approach to software development, where algorithms are manually coded by humans, to ML systems that are developed through data-driven learning. Consequently, the authors must reevaluate our methodologies for software development and address the distinct requirements of these novel system types [63].

2.2 White Shark Optimizer

Population-based metaheuristic solutions play an essential role in solving optimization problems. One of these more contemporary algorithms is considered a promising metaheuristic algorithm: the White Shark Optimizer (WSO) [64]. The White Shark Optimizer (WSO) is a valuable, intelligent metaheuristic model that can resolve a range of issues regarding optimization within an ongoing search area. This 2022 technique mimics how white sharks hunt using their exceptional senses of smell and vision [65]. The White Shark Optimizer (WSO) is an innovative swarm-based metaheuristic algorithm that emulates the predatory behavior of white sharks [66]. This exceptional algorithm offers numerous benefits, such as derivative-free, parameter-less, straightforward, adaptive, reliable, monotonic, sound, and comprehensive. WSO has been employed to address multiple optimization challenges, including the power flow problem [67], power scheduling issues in IoT [68], and path planning for Unmanned Aerial Vehicles (UAV) [69]. WSO, similar to previous MH algorithms, exhibits deficiencies, including slow convergence and imbalanced diversity [68]. Based on the biological traits of white sharks, the following representation of the WSO optimization process is conceivable:

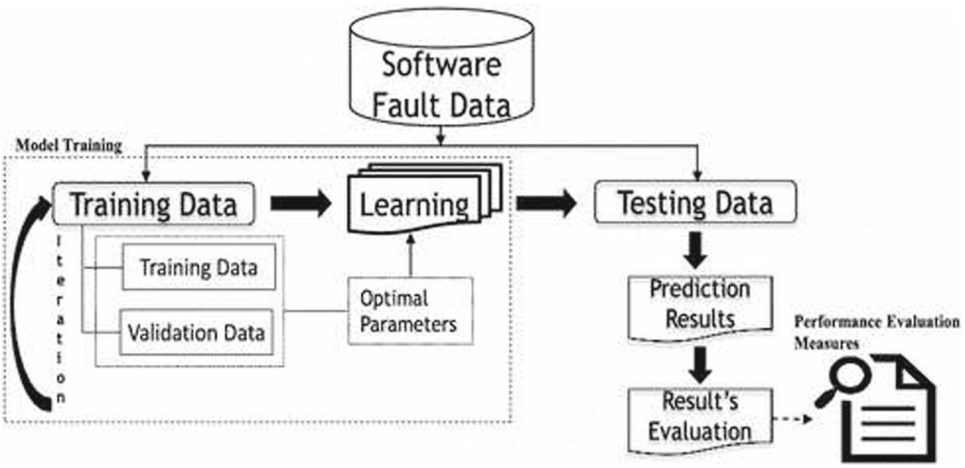

Fig. 2 General software defect prediction process

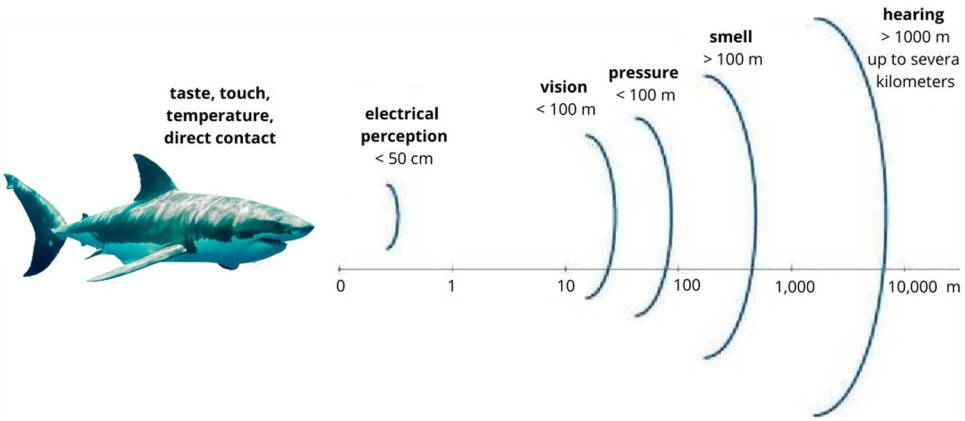

Fig. 3 White shark sense

2.2.1 Inspiration

White sharks, sometimes called great white or white pointers, are among the most potent and deadly predatory sharks worldwide. White sharks are magnificent hunters and highly acclimatized predators. They have strong muscles, vital eyesight for sharp contrast, and an acute sense of smell. Up to 300 obscenely blunt blades and severely pointed, triangle-shaped teeth are placed in multiple rows within their enormous jaws. Their prey include aquatic lions, walrus species, sea turtles, porpoises, small whales, crabs, mollusks, seabirds, and occasionally penguins. Usually, a white shark will ambush its target to take it by surprise before making a sudden, powerful, and lethal bite. White sharks are streamlined torpedo-shaped swimmers with powerful tails that can propel them through the water. They can even emerge from the water and burst like whales, striking prey from below while swimming toward it with undulating movements. The two most fascinating aspects of their collective behavior are how great white sharks acquire meals by swimming and their unique ability to detect and hear the scent of prey.

2.2.2 Tracking the Prey

Like any other organism, white sharks search the ocean for prey, just like any other organism, and adjust their location accordingly. They detect, pursue, and track their prey using nearly all available techniques. As seen in Fig. 3, they have many integrated and complementary senses. First, great white sharks survey a vast area in search of prey thanks to their surprisingly good sense of hearing. Second, they can detect the scent of prey thanks to their keen sense of smell. These characteristics enable them to thoroughly investigate the area and muse every corner of the search arena to locate prey.

2.2.3 Search for prey (exploration)

Using their unusual sense of hearing, great white sharks saunter across the search space in pursuit of prey. As seen in Fig. 4, they are operating two lines on either side of them, allowing them to hear from the entire length of their bodies.

These two lines can detect variations in water pressure, which discloses prey movements. White sharks will become interested in turbulent prey and approach it due to the vari-

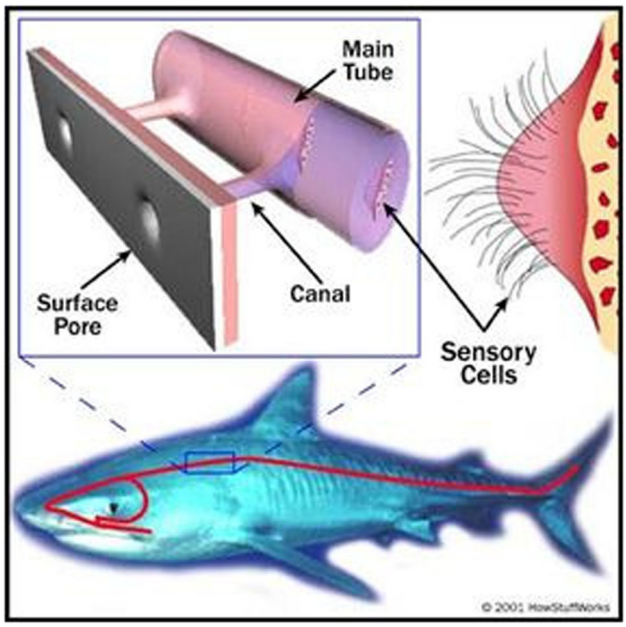

Fig. 4 White shark sense on its torso

ations in water pressure the prey emits. They even possess organs capable of picking up tiny pulses of electromagnetic energy created as prey moves. Then, they can pinpoint the exact location and size of the prey based on the frequency of waves that drift to them during the motion and turbulence of the prey. A white shark can detect electromagnetic fields when it gets this close to its target, moving undulatingly to find its prey.

A formula used to describe the waveform velocity of great white sharks operates as highlighted below

$$v = xf. \tag{1}$$

The number of turns, or cycles, which the white shark finishes in a split second, denoted by f, is the motion's wave frequency that is wavelike. In Hertz (Hz), a cycle per second is measured. Where x is the wavelength, and v is the wave speed. This indicates how far a white shark must travel cyclically to finish a complete turn.

2.2.4 Search for Prey (Exploitation)

With their keen sense of smell, great white sharks use every available place within the realm of space to locate potential prey. A white shark's nostrils are wicked when it gets close to its meal. Amazingly, great white sharks' sense of smell can increase exponentially when they approach their target, allowing them to locate the likely location of the prey precisely. As white sharks tackle their prey, their vantage point can be adjusted using the equation shown below of motion with a constant rate of acceleration

$$x = x_i + v_i \Delta t + \frac{1}{2} a (\Delta t)^2. \tag{2}$$

The latest position of the white shark has been marked by x, its archaic location is shown by x_i, its initial velocity is indicated by v_i, the time gap between its initial and Present positions are embodied by Δt, and The determinant of acceleration, a, is constant. When great white sharks get too close to the scent of prey, they frequently discover no prey there, because the prey, like seals, often leaves their fragrance behind after departing a place. In the present situation, they are forced to randomly look into distinct regions in these arch spaces and inspect other locations using their active sensations of smell, hearing, and sight.

2.2.5 The Mathematical Model of WSO

The projected WSO's mathematical models, created to represent white sharks' behavior as they age, are described in depth in this section. This involves pursuing and monitoring prey.

The deep ocean is home to great white sharks, who can locate prey or food sources. On the other hand, no idea exists regarding where the food supply is situated in a specific search space. In this situation, white sharks must shift through much of the ocean to find food sources. Three behaviors of great white sharks are utilized to find prey or the best place to obtain food: (1) moving in the direction of prey based on wave hesitation brought on by prey movement. In this case, the white shark uses its related senses of smell and hearing to locate prey by swaying; (2) the aimless hunt for food in the ocean's depths. For this reason, great white sharks travel toward their prey's position and remain near the best prey. (3) The white shark's approach to identifying nearby prey. Here, a great white shark approaches the best white shark close to the ideal prey using the schooling behavior of fish. All white shark sites will be updated with the best options if the prey is not identified suitably based on these behaviors. These actions are represented mathematically in the following manner.

2.2.6 Initialization of WSO

Being a population-based algorithm, WSO generates a pool of starting solutions at random to begin the optimization process intended to resolve an issue related to optimization. In a 2D matrix, a potential solution to an issue can be shown as follows: Each white shark's position indicates a possible solution given a d-dimensional search spectrum (i.e., problem dimension) and an estimated number of n white sharks

(i.e., population size)

$$W = \begin{bmatrix} w_1^1 & w_2^1 & \cdots & w_d^1 \\ w_1^2 & w_2^2 & \cdots & w_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_1^n & w_2^n & \cdots & w_d^n \end{bmatrix}. \tag{3}$$

In a given problem, d indicates the number of choice variables, w indicates the exact spot of each white shark in the search space, and w_d^i demonstrates the vantage point of the ith white shark in the dth dimension.

The uniform random initialization method outlined below is used to generate the starting population in the search domain

$$w_j^i = l_j + r \times (u_j - l_j), \tag{4}$$

where r is a random number generated in the interval[0,1], u_j and l_j represent the boundaries of the search space, both upper and lower, respectively, and w_j^i signifies the ith white shark's launching vector in the jth dimension as well.

A fitness function explicitly created for that regard evaluates each potential solution's quality for every new area a white shark chooses to occupy. The former is renovated if the new position proves superior to the existing one. If the white shark's current position is superior to the new one, it stays there in the WSO simulation.

2.2.7 Movement Speed Toward Prey

White sharks dedicate excessive time to searching for and following prey, since they are mammals with an instinct for survival bias. They repeatedly track prey using all available techniques, including their keen senses of smell, sight, and hearing. A white shark swims in an undulating pattern, as shown by Eq. (5), when it uses the hesitation of the waves it hears while the prey is moving to establish its position

$$v_{k+1}^i = \mu \Big[v_k^i + p_1 (w_{\text{gbest}_k} - w_k^i) \times c_1 + p_2 (w_{\text{best}}^{v_k^i} - w_k^i) \times c_2 \Big]. \tag{5}$$

v_{k+1}^i shows the revised vector velocity of the ith white shark, and $i = 1, 2, \ldots, n$ is the white shark population estimate for an identified size. n. In the $(k+1)$th step, v_k^i defines the i-th white shark's speed vector as of right now in the k-th step; w_{gbest_k} represents the newly gained global ideal position vector. thus far by any white shark in the k-th iteration; w_k^i is the white shark's current position vector in the k-th step; $w_{\text{best}}^{v_k^i}$ is the swarm's i-th best-known position vector, and v^i is the i-th index vector of the white sharks' arrival to the optimal location, as shown by Eq. (6). In the interval [0, 1], there are two uniformly generated random numbers, c_1 and c_2. The forces of the white sharks, denoted by p_1 and p_2, govern the influence of w_{gbest_k} and $w_{vi\text{best}}$ on w_i^k, respectively. These forces are calculated using the formulas provided in Eqs. (7) and (8). The constriction factor μ, defined as present in Eq. (9), is proposed by WSO to study the convergence behavior of white sharks

$$v = \lfloor n \times \text{rand}(1, n) \rfloor + 1, \tag{6}$$

where rand(1,n) symbolizes the vector of arbitrary numbers distributed regularly in the interval [0,1]

$$p_1 = p_{\max} + (p_{\max} - p_{\min}) \times e^{-\left(\frac{4k}{K}\right)^2} \tag{7}$$

$$p_2 = p_{\min} + (p_{\max} - p_{\min}) \times e^{-\left(\frac{4k}{K}\right)^2}. \tag{8}$$

For white sharks, the initial and secondary velocities requisite for sustaining good motion are depicted by p_{min} and p_{max}, where the momentary and highest possible number of iterations are denoted by the two numbers k and K, accordingly. Following a thorough investigation, the values of p_{min} and p_{max} were determined to be, respectively, 0.5 and 1.5

$$\mu = \frac{2}{\left| 2 - \tau - \sqrt{\tau^2 - 4\tau} \right|}, \tag{9}$$

where the acceleration coefficient is indicated by τ.

2.2.8 Movement Toward Optimal Prey

Most of a great white shark's time is spent hunting for possible prey, where an ideal or suboptimal meal may be found. As a result, the white sharks' positions are constantly shifting. Usually, they will approach prey when they smell or hear the waves created by the prey's movement. Sometimes, the prey wanders away from its original site in search of food or when a white shark approaches it. In such a position, the prey frequently leaves its scent behind whenever the white shark still notices the victim. In this instance, the white shark navigates randomly to find prey, akin to a group of fish scavenging for food. In the present scenario, the position update method provided in Eq. (10) illustrates how white sharks respond when they get closer to prey

$$w_{k+1}^i = \begin{cases} w_k^i \cdot \neg \oplus w_o + u \cdot a + l \cdot b & \text{if rand} < m_v \\ w_k^i + \frac{v_k^i}{f} & \text{if rand} \geq m_v, \end{cases} \tag{10}$$

where l and u correspondingly reveal the search space's upper and lower bounds, $aandb$ are one-dimensional binary vectors

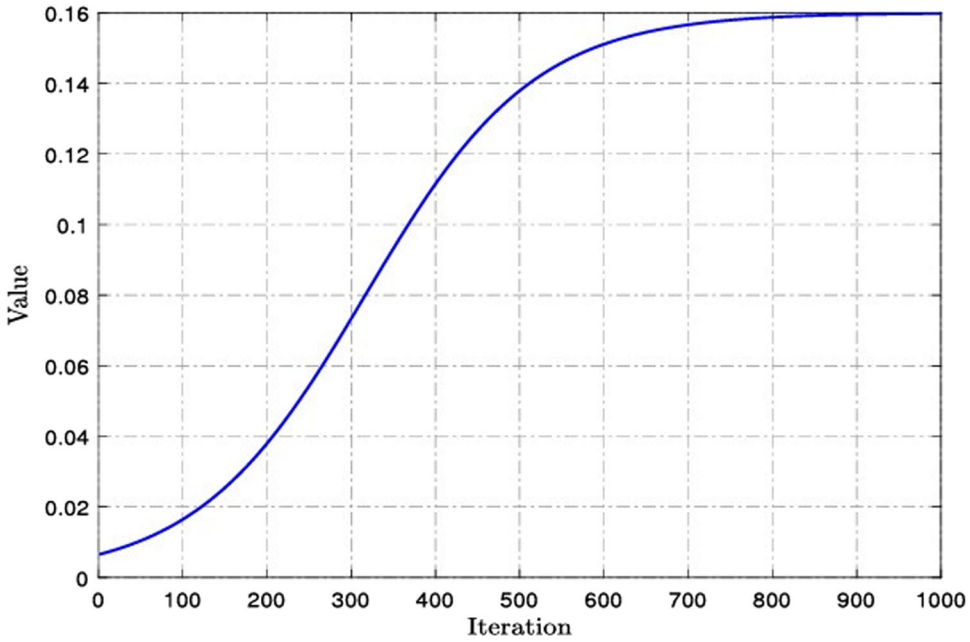

Fig. 5 An iteration-based function that depicts the overall growth trend in the senses of hearing and smell used by white sharks for spotting prey

defined by Eqs. (11) and (12), and the ith white shark's newly acquired position vector in the $(k+1)th$ iteration step will be indicated by w_{k+1}^{i}. A logical vector denoted by w_0 is defined as described in Eq. (13), f signifies the frequency of a white shark's wavy behavior, and Eq. (15) defines the movement force that rises as the shark gets closer to its prey. Rand is a term used to describe a random number produced between 0 and 1

$$a = \text{sgn}(w_k^i - u) > 0 \tag{11}$$

$$b = \text{sgn}(w_k^i - l) < 0 \tag{12}$$

$$w_o = \oplus(a, b). \tag{13}$$

For which the bit-wise xor operation is $\oplus$. Equations (11) and (12) are crucial to help white sharks explore every possible area of the search space and to support solutions that act arbitrarily in the search space

$$f = f_{min} + \frac{f_{max} - f_{min}}{f_{max} + f_{min}}, \tag{14}$$

where rand is a random number evenly distributed throughout the scope, and f_min and f_max indicate the minimum and maximum frequencies of the undulating motion, respectively[0,1]

$$mv = \frac{1}{a_0 + e(\frac{k}{2} - k)/a_1}. \tag{15}$$

Two positive constants, a_0 and a_1, are employed to oversee the actions of exploration and exploitation. The power of the white shark's senses of smell and hearing was proposed to be expressed by the parameter mv, which grows with the number of iterations. This function is drawn over throughout iterations, as seen in Fig 5.

2.2.9 Movement Toward the Best White Shark

Great white sharks can hold their position about the best nearby prey. Equation (16) is the formulation of this behavior

$$\omega_{k+1}^{\prime i} = \omega_{gbest_k} + r_1 \overrightarrow{D}_{\omega} sgn(r_2 - 0.5) r_3 < S_s. \tag{16}$$

The ith white shark's revised position about the prey's position is ω_{k+1}^{i}. sgn$(0.5 < timesr_2)$ returns either 1 or -1 to reverse the search's direction. The random numbers that make up the variables r_1, r_2, and r_3 are in the interval [0, 1]. $\overrightarrow{D}$, as stated by Eq. (17), is the space separating the food supply or victim and the white shark. As demonstrated through Eq. (18), S_s is a metric developed to characterize the degree to which white sharks can see and smell whenever they proceed to other white sharks adjacent to prospective prey

$$\overrightarrow{D}\omega = |\text{rand} \times (\omega_{\text{gbest}_k} - w_k^i)|. \tag{17}$$

The white shark's current position about ω_{gbest_k} is represented by w_k^i, and a rand is a random number ranging from zero to one

$$s_s = |1 - e^{(-a_2 \times k/K)}|. \tag{18}$$

The positive constant a_2 regulates the exploration and exploitation tendencies.

2.2.10 The Fish School Mentality

The top two responses were retained, and the positions of additional white sharks were updated based on these ideal placements, creating a mathematical simulation of how the white shark school behaved. It was suggested that the following formula describe how white sharks behave in schools of fish:

$$\omega_{k+1}^{i} = \frac{\omega_{k}^{i} + \omega'^{i}_{k+1}}{2 \times rand}, \tag{19}$$

where the symbol rand specifies a random number with a uniform distribution inside the range of [0,1] [65, 67, 70–74]. The overall representation and discussion of the WSO algorithm is shown in 1.

2.3 Binary White Shark Optimizer

The White Shark Optimizer (WSO) is a metaheuristic algorithm inspired by white shark hunting behavior, especially their techniques for identifying, striving for, and attacking prey [65]. The White Shark Optimizer (BWSO) modifies the technique for binary optimization problems with binary variables (0 s and 1 s). The fundamental ideas of Binary White Shark Optimizer are mentioned below:

- Binary Representation: The binary version represents solutions as binary strings. Each string element can be 0 or 1, signifying various states or options in the problem space.
- Search Strategies:
  - Exploration: During the early stages, BWSO prioritizes exploration, encouraging varied solutions by replicating sharks' random quest for prey throughout the vast ocean.
  - Exploitation: As the search advances, BWSO eventually moves toward exploitation, focusing on fine-tuning the best solutions discovered, similar to sharks' precision in pursuing prey.
- Position Update Mechanism:
  - Continuous optimization techniques often use mathematical formulas to update positions (solutions). However, in BWSO, position updates are motivated by probability-based methods determining whether each bit in the solution string should be flipped (from 0 to 1 or vice versa).
  - This process may employ the sigmoid function or other transfer functions to convert the continuous search space into a probability that regulates the binary update.
- Fitness Evaluation: Each candidate solution's fitness is evaluated using the problem-specific objective function, which guides the BWSO's search process.
- Dynamic Positioning: The program uses mathematical rules to alter the positions of candidate solutions, simulating white shark movement and hunting activity. This technique relies on dynamic interactions throughout the population rather than a single leader.
- Attack Phase: In the final stage of the algorithm, BWSO speeds up the search for potential regions by imitating the shark's last attack on prey. This phase improves exploitation by refining the finest solutions uncovered throughout the search process.

Figure 6 shows the binary white shark optimizer process.

The WSO is updated and applied in this work to handle the software defect classification problem. Furthermore, the WSO's optimization techniques are adjusted to account for the binary nature of this issue. The WSO was chosen for adaptation because of its superior exploring skills and efficacy. The results of the proposed WSO are compared to those of the Firefly, Harris Hawks Optimization (HHO), Cuckoo Search (CS), and Particle Swarm Optimization algorithms (PSO). The recommended Binary White Shark Optimizer (BWSO) surpassed the others in terms of accuracy and optimal fitness values.

3 Literature Review

In addition to a section devoted to reviewing studies of the WSO optimizer and its variants, the authors of this article will address the most recent research on software defect prediction using various ensemble learning techniques in this section.

3.1 Predicting Software Defects Using Ensemble Learning Methods

Software defect prediction (SDP) detects defective software components to improve project quality and reduce maintenance risks. SDP links software metrics and defects via multiple methods using past defect data. Many methodologies and frameworks have been described to discover software module faults using machine learning (ML) and deep learning (DL) prediction models. These binary classification models struggle most with class imbalance. When there is unequal class distribution, accuracy may be good, but models cannot discriminate minority class data samples, resulting in bad classifications. Earlier research on SDP class inequality has been limited. Data sampling is employed to solve the class imbalance problem and improve this study's SDP ML model performance. A convolutional

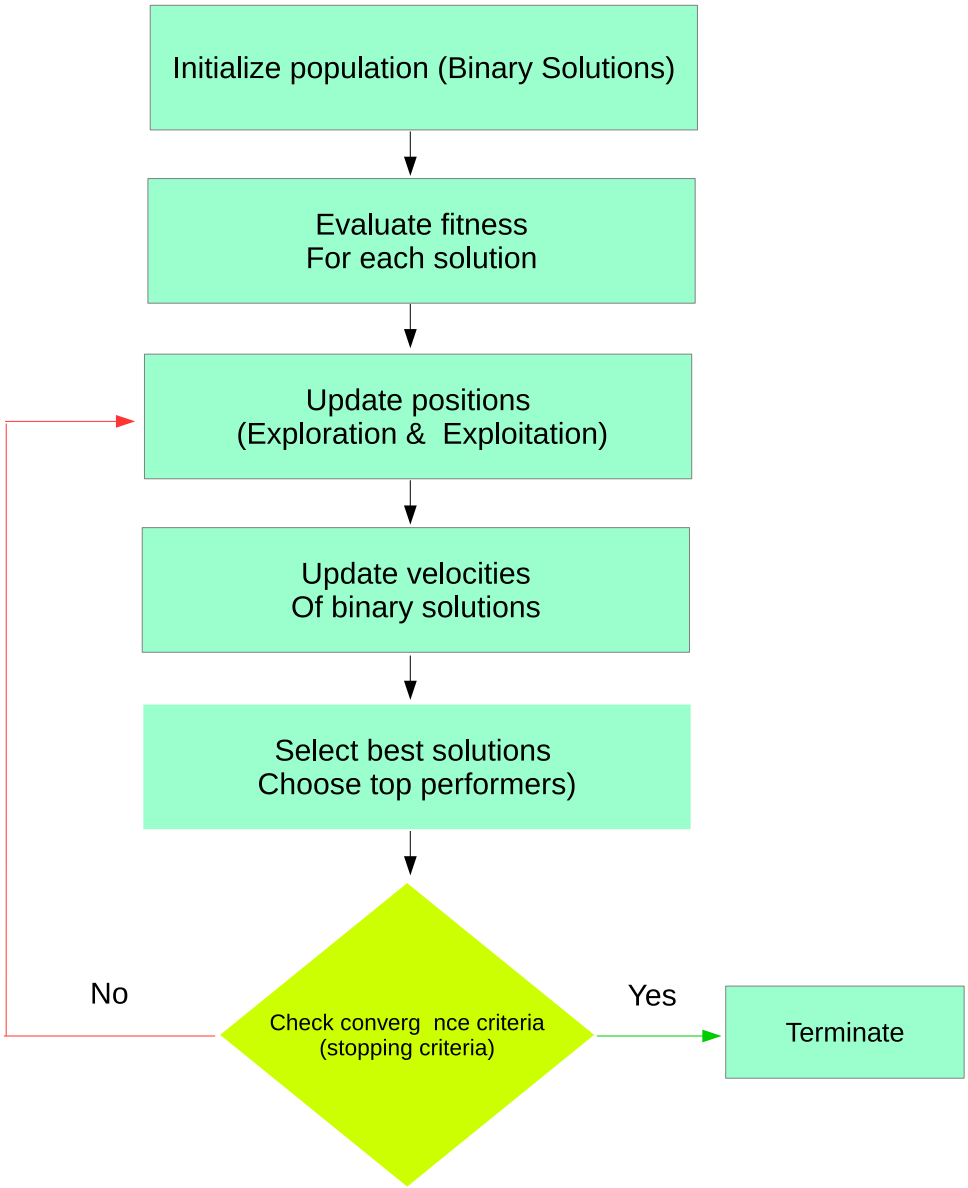

Fig. 6 Binary white shark optimizer flowchart process

neural network (CNN) and gated recurrent unit (GRU) with synthetic minority oversampling and the Tomek connection predict software defects. The models' efficiency was tested using PROMISE repository benchmark datasets. Experimental findings were compared and interpreted using accuracy, precision, recall, F-measure, Matthew's correlation coefficient, area under the curve, precision–recall curve area, and mean square error. Tests indicate that the proposed models accurately forecast software faults on balanced datasets, with a 19% CNN and 24% GRU AUC improvement. The authors compared the SDP technique using common performance measures. The proposed strategy outperformed the conventional SDP algorithms on most datasets [75]. As software technology has advanced, many complex applications have evolved in various industries. Businesses, in particular, rely on software-based applications to provide cutting-edge services. However, error prediction in software is a significant barrier that industries must address to promote corporate success. As a result, new strategies for predicting faults at an early stage of the software life cycle are required to eliminate software flaws later on. Diverse automated defect prediction solutions tackle the difficulties associated with manual forecasting. All possibilities employ pattern recognition to identify software issues through associated characteristics. Notwithstanding the existence of defect identification technologies, enhancing performance remains challenging. The authors present an efficient hybrid machine learning approach for software failure prediction to address the limitations of current prediction methods. The initial section enhances dataset features with a genetic algorithm (GA)

that selects attributes based on an improved fitness function. Upon identifying the optimal characteristics, the Decision Tree method categorizes them. The research contrasts the GA-DT-based hybrid model with RCSOLDA-RIR and WPA-PSO for the prediction of software failures. The experimental study demonstrates that the proposed model exhibits greater accuracy than the existing one [32]. A newly developed correlation-based modified long short-term memory neural network (CM-LSTM) is used to predict software errors in software projects based on modeled data. The intended variables were changed due to the positive relationship between features and target factors. The prepared data are sent into the LSTM model to correct an imbalance in the data for software defect predictions. A JM1 software defect prediction dataset with various performance criteria is utilized to evaluate the suggested technique. The improved correlation-based LSTM approach is good at detecting software flaws. To discover software faults more effectively than correlation-based LSTM, KNN, stochastic gradient descent, RF, NB, LR, DT, LDA, MLP, and others, the suggested method leverages correlation-based feature selection for long short-term memory neural networks [37]. Researchers [76] present their findings on ensemble techniques for SFP. Rotation Forest, Dagging, Decorate, Grading, MultiBoostAB, Real-AdaBoost, and Ensemble Selection are the seven ensemble strategies that the authors empirically assess. In our opinion, most of these ensemble techniques have never been used with SFP. Using the benchmark fault datasets, researchers run multiple investigations deploying three classification methods as base learners for the ensemble approach: naive Bayes, logistic regression, and J48 (decision tree). Based on the experimental analysis, Decorate had the highest AUC value (0.986) and the maximum precision, recall, and G-mean 1 values (0.995, 0.994, and 0.994, respectively) while rotating forest with J48 as the base learner. Furthermore, the results of the statistical tests revealed that the performance of the ensemble techniques used for SFP was satisfactory. Additionally, the cost–benefit analysis demonstrated that for 20 of the 28 used fault datasets, SFP models based on employed ensemble techniques might be beneficial in reducing software testing expenses and labor.

Wang et al. [77] propose addressing the issue of low prediction accuracy in most SDP models. A proposed SDP model combines the support vector machine technology with the least absolute value compression and selection approach. First, the dimension of the original data set is reduced using the feature selection capability of the minimal absolute value compression and selection approach, and the SDP-unrelated data set is removed. The cross-validation algorithm's parameter adjusting ability is then employed to obtain the optimal SVM value. Finally, SVM's nonlinear computing capabilities round out the SDP. The results show that the defined defect prediction model outperforms conventional models regarding prediction speed and accuracy. To tackle the problem of class disparity, a K-nearest neighbor (KNN) filtering-based data preprocessing strategy for stacked ensemble classifiers is proposed. To lower the unbalanced ratio, overlapped data points are eliminated using closest-neighbor-based filtering. After processing, the stacked ensemble receives the data with static code metrics to make predictions. The stacking uses five base classifiers: Support Vector Machine, K-nearest neighbor (KNN), Decision Tree, Naive Bayes, and Artificial Neural Network. Thirty classifiers are compared (5 prediction strategies * 6 data preprocessing procedures). The research uses five publicly available datasets from the NASA repository: PC1, KC1, KC2, JM1, and CM1. One hundred and fifty prediction models are proposed, five for preprocessing data, six for classification, and five for datasets. Each model is evaluated for performance using the accuracy, area under the curve, and receiver operator curve. Regardless of the datasets, the statistical analysis reveals that the recommended stacked ensemble classifier with KNN filtering works better than any other predictors [78].

The researchers' two main tasks to enhance performance are variant-based ensemble classification and feature selection. In addition to lowering the framework's performance due to high processing costs, feature selection removes features not involved in the classification process. Before the variant selection procedure, six base classifiers, SVM, DT, KNN, NB, RF, and MLP, are optimized to construct the variations. One version is chosen from each basic classifier variant (classifier family) based on how well it outperformed all other variants in the family, including the base classifier. Three variants, SVM-4, RF-3, and KNN-4, with high performance, are chosen for this study. After that, these variations are combined using an ensemble technique called "voting" with every potential combination. Outperforming all other combinations, RF-3 and KNN-4 are chosen for classification in the suggested framework. The framework's output is contrasted with several popular supervised classifiers from published papers that performed performance analysis using the same datasets and performance metrics. The comparative analysis revealed that the suggested framework eliminated the class imbalance problem and performed better than any other classification method from the published research. To pick additional variants for ensemble learning, it is recommended that future studies optimize more classifiers with a wide range of parameters [79].

Using the jedit4.0, ant1.7, and camel1.4 datasets, respectively, the authors have examined the factor of classification accuracy in three scenarios in this study. In the first example, the accuracy gained by applying the suggested bagging method on JEDIT 4.0 was 96.7, while random forest yielded an accuracy 91.5. These perform far better than alternative machine learning techniques. Next, the suggested approach outperformed the other machine learning models with an

accuracy 96.2 in the ant1.7 dataset. In the final instance, the proposed strategy surpassed others with an accuracy of 95.9 using camel 1.4. In addition to the accuracy measure, the suggested method performs noticeably better than all machine learning methods in terms of TPR, FPR, precision, TNR, F-measure, and AUC-ROC. The analysis demonstrated that the bagging classifier performs exceptionally well across all datasets and performance metrics [80].

In the [81] study, the authors base the analysis on software failure prediction using a modified stacking ensemble of tree-based ensembles. There are two primary contributions from this study: (1) Hyperparameter optimization was utilized for a total of seven tree-based ensembles to examine how it impacts ensemble models; (2) to determine the extent to which it would enhance the prediction performance over fine-tuned tree-based ensembles, the authors built a stacking ensemble of fine-tuned tree-based ensembles. The study's findings demonstrate the significant influence that hyperparameter optimization has on additional trees and random forest ensembles. Furthermore, compared to all other fine-tuned tree-based ensemble models, the stacking ensemble of these ensembles demonstrated excellent prediction performance.

Using a combination of Adaboost, Random Forests (RF), and Naive Bayes, the Voting-Based Ensemble Learning classifiers are used in this work [82] to construct a prediction model of software flaws. Twelve NASA datasets are utilized following feature selection on the pre-manipulated data utilizing a heuristic-based approach and a wrapper-based approach. Regarding functionality, wrapper-based previously processed information performs better than heuristic-based cleaned data. The precision, f-score, AUC, and recall values are finally determined. Based on the vast majority of the datasets, it can be concluded that the Voting-Based Ensemble model exceeds other AI models. One disadvantage is that the minimal testing sets of the KC3 and MC2 datasets may impact their accuracy.

Predicting a module's propensity for defects is possible, which can save time, money, and labor when developing a software project. Although identifying the underlying causes of software errors can be difficult, many machine learning models continue to be researched to offer very successful prediction systems. To help with the prediction model's poor classification rates, a hybrid approach known as the diverse ensemble learning technique (DELT) is suggested as a solution for the within-project defect prediction (WPDP) dilemma. DELT used two different perspective-generating methodologies, namely, bootstrap aggregation and multi-inducer. The proposed DELT uses majority voting to anticipate the eventual class label for every unlabeled test module. 43 NASA and PROMISE datasets accessible to the public are used for various activities. The experimental results show promise, since they enhance the software module's ability to classify defects more accurately through generalization [83].

In this paper [84], authors provide a sequential-ensemble model to forecast software errors. The use of ensemble modeling in the prediction of software faults is driven by its versatility. In addition, the eight datasets from the PROMISE and ECLIPSE repositories are used to test the suggested methodology. The average absolute error, average relative error, and prediction are the three error metrics used to measure the performance of the proposed model. The positive outcomes demonstrate the effectiveness of the suggested model.

This work [85] offers two new methods for learning from unbalanced data sets that outperform the minority class in terms of prediction accuracy. These two approaches are effective at handling various parts of unbalanced classification. Whereas the other manages data sets that are moderately imbalanced, the first one deals with very imbalanced sets. Their differences lie in whether or not they employ expenses for misclassification and oversampling during the training phase. The findings from all of the experiments prove that the two of these techniques have achieved outstanding results in both G-mean and AUC measures and have precisely identified the defective modules, thus lowering the cost of the detection system compared to other state-of-the-art imbalance learning algorithms on imbalanced datasets.

Most recent research concentrates on automating this process from various angles, like determining the bug's importance or severity. They neglected to take into account the fact that the defect is a multi-class classification challenge, though. This paper resolves this issue by putting out a novel prediction model to examine BRs and forecast the type of bug. Using machine learning and natural language processing (NLP) methods, the suggested model builds an ensemble machine learning algorithm. A dataset for two online software bug repositories that are freely obtainable (Mozilla and Eclipse) is used to simulate the suggested model. The dataset comprises six classes: Program Anomaly, GUI, Network or Security, Configuration, Performance, and Test. According to the simulation findings, the suggested model can outperform the majority of current models in terms of accuracy, achieving 96.72 with text augmentation and 90.42 without it [86]. Table 1 gives an overview of the research done on the Ensemble method between 2021 and 2024.

3.2 Evolutionary Algorithms in Software Defect Classification

Optimization is an organized approach that aims to maximize or minimize a predefined objective function by modifying specific parameters to find the most effective use of resources or the best solution to a problem given a set of constraints and assumptions. It evaluates the objective functions to select the

Table 1 SDP using ensemble learning methods through the years 2021–2024

Year	Refs.	Technique	ML model	Evaluation parameters	Advantages	Disadvantages	Limitations
2024	[87]	Dual ensemble SDP (DE-SDP)	NN	ROC curve (AUC), F1 score, MCC	Utilizes the entire dataset for maximum efficiency	Limited to WPDP	Considers only traditional metrics
	[88]	Intelligent ensemble-based SDP	RF, SVM, NB, ANN	TP, TN, Accuracy, FP, FN	Outperformed modern research	Performance varies with training datasets	Missing data impacts predictions
	[89]	SMOTE	SVM, NB, RF, KNN, MLR, J48	Accuracy, Precision, Recall, F1	Assists in bug severity assignment	–	Supervised learning only
2023	[90]	Soft Voting, Stacking, Gradient Boosting, Bagging	RF, XGb, Adaboost, etc	Accuracy	Improved results through preprocessing	–	–
	[50]	Bagging ensemble (AVSEB)	ELM, SSA-ELM, etc	Recall, F-measure, G-mean, MCC	Higher accuracy and generalization	Time-consuming parameter selection	Explore efficient algorithms further
	[91]	Ensemble Logistic Regression	Various models	MAE, RMSE, TPR, F1, AUC	Efficiently classifies buggy software	–	–
	[92]	Three ensemble learning	J48, RF, RepTree, NB	Precision, Recall, ROC, F1	Surpasses traditional techniques	–	Investigate closed-source applications
	[31]	Twin support vector machine	SVM, TWSVM, etc	Accuracy, Precision, F1, etc	Maximizes interclass spacing	–	Effective parameter selection is vital
2022	[93]	Random approximate reduct (RAR)	CART, KNN	F1, MCC, Recall	Perturbs attribute and instance spaces	–	Oversampling or undersampling needed
	[83]	Majority voting	Various models	Accuracy, AUC	Better performance with diversity	–	Costly to create and train
	[94]	Nested-Stacking	Stacking, TCA, DBN	AUC, F1	Excellent classification performance	–	–
2021	[80]	Bootstrap aggregating	Various models	Accuracy, Recall, etc	Improves classification accuracy	–	Requires deep learning
	[82]	Voting-Based Ensemble	Various models	Precision, Recall, F1, AUC	Better for defect prediction	–	Small test sets may affect accuracy
	[78]	Stacked ensemble classifier	Various models	AUC, Accuracy	Efficiently handles class imbalance	–	Replicable with larger datasets

best option from various viable alternatives. Most situations, such as finding the fastest path between two points, increasing equipment output in a plant, or optimizing machine usage to reduce downtime, can be described as optimization problems. The main goal of any optimization software is to solve an optimization problem with desirable results. Thus, optimization is desperately needed in many domains, including engineering problems [95], feature selection [96, 97], COVID-19 prediction [98], chemo-informatics [99], and image segmentation and threshold [100].

Software testing is crucial to software development and often determines project success. Although crucial, current projects' fast pace and tight deadlines often lead to them being neglected or not comprehensive enough due to a lack of time, resulting in the loss of reputation, private users' data, money, and even life. In such instances, predicting which modules are error-prone based on software data and focusing testing on them is crucial, a common classification challenge. Machine learning models have been successful in many classification tasks, and this paper presents the eXtreme gradient boosting (XGBoost) model for defect prediction. The XGBoost hyperparameters can be calibrated using a modified reptile search optimization method. HARSA, the improved technique, performed well on tough CEC2019 benchmark functions. The suggested algorithm-based XGBoost model was then tested on two benchmark software testing datasets and compared to other sophisticated swarm intelligence metaheuristics in the same experimental context. Both datasets were classified better using the proposed method. Finally, the Shapley Additive Explanations investigation examined how software metrics impact classification results [101]. Software quality is important, because it is employed in many applications. Software defect prediction (SDP) fixes bugs and boosts performance. Existing SDP methods prioritize robustness and dependability. This research introduces a hybrid optimization-based neural network (Optimized NN) for software defect detection. Optimization NN-based SDP focuses on feature selection and SDP with Optimized NN. The relief algorithm finds faults and no-defects features in the feature selection module. The SDP module receives features, and the hybrid optimization of the social spider algorithm (SSA) and gray wolf optimizer (GWO) tunes the NN classifier optimally. The suggested prediction model achieved 93.64% accuracy, 95.14% sensitivity, 99% specificity, 93.53% F1-score, and 99% precision for K-folds [102]. Software fault prediction (SFP) is the early detection of fault-prone modules in software development that are prone to failure and entail significant development costs. It is essential to ensure a high-quality finished product. Machine learning-based classifiers are widely employed for SFP. The Curse of Dimensionality threatens classifier performance in predicting fault-prone software modules. This paper examines the metaheuristics for selecting the best feature subset from a high-dimensional defect dataset. The article offers a Lion Optimization-based Feature Selection (LiOpFS) model and statistically compares it to current metaheuristic models. The NASA dataset was used for the experimental investigation. The results show that the LiOpFS algorithm outperforms baseline approaches, with the highest AUC (90.1%) and Accuracy (94.2%). The results are statistically validated using the Friedman Test with a confidence level of 95% [103]. When there are defects in the software, it takes more time and money to complete the project and distribute the finished product. Defect monitoring and repair are software procedures that can be expensive and time-consuming to finish. Since finding and fixing every flaw in a product is impossible, reducing their detrimental effects is crucial to producing a higher quality final product. Software defect prediction is the process of locating problematic areas of software code. To enhance software quality, an optimal machine learning-enabled model for software defect prediction is presented in this research. This model uses the PC1 data set as its input data. The approach of ant colony optimization (ACO) is utilized to pick significant traits. The support vector machine receives input consisting of specific features. The PC1 data set is used to train and evaluate SVM. ACO SVM's performance SVM, Naive Bayes classifier, and K-Nearest Neighbor classifier are compared with the Ant Colony Optimization Support Vector Machine. ACO-based SVM performs better for software fault prediction and classification [104]. Problems arise due to software defects, mistakes, and bugs. Software problems arise from incorrect requirements, flawed architecture, and faulty source code. Software engineers can speedily find and fix bugs using a few different ways. A few high-quality feature subsets can be extracted from any dataset. Choosing the right attributes can help with classification in a roundabout way. The GJO algorithm, a metaheuristic optimization method inspired by golden jackal hunting, is utilized in a novel feature selection (FS) approach. The software failure prediction dataset features are selected using this method, which employs K-Nearest Neighbor, Decision Tree, Quadrative Discriminant Analysis, and Naive Bayes. To prove this strategy, the Authors shall compare the genetic approach, ant colony optimization, particle swarm optimization, and derivative evolution. FSGJO was usually successful. Classification accuracy was improved for numerous outcomes by the FSGJO. The new approach achieved superior quality selection, according to Friedman and Holm tests [105]. The optimization approach (search strategy) and assessment criteria (objective function) compose an optimization issue solution. An extended binary version of the Harris Hawk Optimization method (EBHHO) is utilized to find a (almost) optimal solution to the Feature Selection (FS) problem in this [106]. As evaluation criteria, K-nearest neighbors (kNN), Decision Trees (DT), and Linear Discriminant Analysis

(LDA) classifiers were used to build the objective function. Adaptive Synthetic (ADASYN) oversampling was used to rebalance the dataset and improve the learning process after FS lowered its dimensionality. The proposed method was tested using well-known Software Fault Prediction datasets. A study showed that EBHHO is better than HHO. It was shown that the EBHHO algorithm outperformed other optimization methods.

This study uses an optimization technique to enhance the overall results. The optimizer is called White Shark Optimizer (WSO).

3.3 White Shark Optimizer (WSO)

First released in 2022, the White Shark Optimizer is a novel algorithm that mimics the hunting habits of white sharks and is inspired by nature. The benefits of the WSO are its robust durability, high adaptability, and simplicity. However, it also has several drawbacks, including a tendency to reach the regional optimum, a small search radius, and a lack of demographic variety [107].

A novel bio-inspired metaheuristic algorithm is presented to address a range of global optimization problems. The main goal of the WSO algorithm is to simulate the White sharks' unpredictable behavior, including their foraging and navigating hearing and smell activities. Three stages of movement, moving quickly toward prey, moving toward ideal prey, and moving toward the best white shark, are necessary for the optimal outcome in the WSO. Fish school behavior is also taken into consideration [64].

The remarkable senses of smell and hearing possessed by great white sharks, which are essential to their hunting and mobility, are the foundation of this algorithm. The authors numerically model and quantitatively analyze unique characteristics to balance scheme use and research to help search agents investigate and use all likely zones in the search region for better optimization. WSO search agents canify their placements to apprtost answers and effectively accomplish the intended results [108].

A White Shark Optimization (ELWSO) Algorithm Based on Elite Opposition is suggested by [109] to solve the issue of where to place uncrewed aerial vehicles (UAVs) in smart cities. Using the Elite opposition-based technique, the EWSO scheme improves the optimization efficiency of the original WSO. Based on fitness, coverage, and connection criteria, EWSO was assessed in 23 scenarios with different numbers of users and UAVs. The EWSO approach outperformed the WSO, Genetic method (GA), Bat Algorithm (BA), and Particle Swarm Optimization (PSO) in the simulated trials conducted with MATLAB 2021b.

The Balance of Fitness Distance in Roulette, the White Shark Optimization technique, debuted in [110], is recommended as a solution to the mesh router location with service priority problem. The efficacy of the proposed method is verified by optimizing the fitness value to improve coverage and connection according to the weights and locations of a particular client group. The outcomes of the generated runs indicate that the suggested approach far surpasses the original versions of White Shark Optimization, Particle Swarm Optimization, Genetic Algorithm, Sine Cosine Algorithm, and Harmony Search when it produces competitive results.

In [111], a more sophisticated multi-objective white shark algorithm (MOWSO) based on Levy flight, Cauchy mutation, and nonlinear weight factor is presented. This upgraded MOWSO aims to reduce the costs associated with the system's economic operation and environmental remediation. The improved method's efficacy is illustrated through the multi-objective test function, and an example is used to solve the optimization model. The simulation's results show that the model can successfully reduce operating and environmental expenses while maintaining the economic operation of a microgrid.

Amor et al. [72] examine how a recently created metaheuristic algorithm influences the surface roughness and machinability (cutting force) of glass fiber-reinforced polymer composites with incorporated nano zinc oxide (nZnO-GFRPC). The grey theory merges the force of cutting and roughness of the surface output answers into a unary objective function in a hybrid grey theory-white shark optimizer (Grey-WSO) algorithm. In contrast, the white shark is utilized to identify the best replies. The presented method is innovative in combining two distinct responses, namely cutting force and surface roughness, into a single objective function and integrating two machine learning algorithm kinds into one. Grey-WSO is used to construct the Taguchi orthogonal array and optimize several parameters, including feed rate, fiber volume fraction, and number of nanoparticles. The best results are achieved at 1% ZnO (Weight%), 75 mm/min feed rate, and 6.031% fiber volume fraction, respectively. The results showed that the optimal cutting force and surface roughness were 1.6765 μ and 197.64 N, respectively. The output performance increased from 0.9414 to 0.9514, according to the validation of the results, showing that the created Grey-WSO performed with a 1.06% error. It was compared to existing metaheuristic algorithms to show the created algorithm's potential for use in composite materials' shaping, cutting, milling, and other machining properties. The outcomes further demonstrate that the quantity of nanoparticles significantly impacts surface roughness calculations.

HSlopEn and support vector machines (SVM) are optimized using the white shark optimizer (WSO); WSO-HSlopEn and WSO-SVM are recommended. Next, a dual-optimization fault diagnostic method for rolling bearings is suggested. It is based on WSO-HSlopEn and WSO-SVM. First, various bearing signal types are input and split into mul-

tiple nodes using hierarchical decomposition. Then, using the recognition rate as the fit function, WSO adaptively determines the HSlopEn and SVM parameters. Subsequently, the nodes' WSO-HSlopEn is retrieved, and both single- and multi-feature extraction are carried out. WSO-SVM finally outputs the diagnosis results. The results of the experiment showed that the WSO-HSlopEn and WSO-SVM fault diagnosis method, whether used in single- or multi-feature settings, has the highest recognition rate in contrast to other hierarchical entropies; additionally, in multi-feature settings, all recognition rates exceed 97.5%, and the recognition effect improves with the number of features selected. When five nodes are chosen, the highest recognition rate is 100% when five nodes are chosen [112].

The technique of estimating the likelihood that a business may experience bankruptcy or other financial difficulties in the future is known as bankruptcy prediction. Creditors, investors, and financial institutions can evaluate credit risk, make wise investment decisions, and implement suitable risk management strategies using an accurate bankruptcy prediction model. Numerous approaches, including conventional statistical methods and more sophisticated machine learning (ML) techniques, have been developed to consider bankruptcy. To estimate the likelihood of bankruptcy, this method often uses financial ratios, accounting data, market performance indicators, and other relevant variables as input features. Since deep learning (DL) techniques became popular, there has been an increasing interest in using neural networks to predict bankruptcy. This article presents the WSODL-BPFCA approach, a new white shark optimizer that uses deep learning-based bankruptcy prediction for financial risk assessment. The proposed WSODL-BPFCA method predicts the presence of bankruptcy using a DL model that has been hyperparameter-tuned. The WSODL-BPFCA method uses min–max normalization to convert the input data into a uniform format. The WSODL-BPFCA technique presents an attention-based long short-term memory (ALSTM) strategy for bankruptcy prediction. Finally, the WSO technique adjusted the ALSTM model's hyperparameters. Many simulations were run to demonstrate the improved performance of the WSODL-BPFCA approach. The thorough comparison analysis showed that the WSODL-BPFCA technique produced improved outcomes, with a 97.61% improvement in several criteria [113].

The Internet of Things, or IoT, is a network of interconnected devices that can communicate with each other and share data because of the Internet. Smooth data output is critical to a network's lifetime, and wireless sensor networks (WSN) are a significant component of the Internet of Things in this respect. Despite the benefits of the Internet of Things, security, energy, load balancing, and storage remain significant challenges. Two techniques utilized within the structure of an IoT-assisted WSN to reduce energy usage are multi-hop routing and clustering [114]. This offers a brand-new, highly successful hybrid optimization technique for selecting cluster heads. The proposed method uses the whale optimization approach (WOA) to modify the white shark optimizer's (WSO) stochastic behavior as it searches for food. Modern metaheuristic techniques, including the artificial optimizer (GTO), the coyote optimization algorithm (COA), and the original WSO, were also tested against the new HWSO. Finally, the complete simulation features of NS-3.26 are used to validate the suggested network. The simulation results may demonstrate improvements in the packet delivery ratio (PDR), latency, energy usage, the number of deceased nodes, and network durability.

The growing number of users is the reason for the rising demand for virtual machine (VM) requests. Therefore, in cloud data centers (DCs), virtual machines (VMs) are crucial for efficient resource management. The set of virtual machines (VMs) is generally deployed into the set of physical machines (PMs) via the VM placement procedure based on predetermined criteria [73]. Hybrid optimization using fitness parameters determines the best location for virtual machines. The migration expense, placement time, power, and load objectives are combined to compute the fitness function and several system parameters. When arranging virtual machines (VMs), factors, such as the processing elements, memory, bandwidth, CPU, and million instructions per second (MIPS), are considered. Further, the hybrid optimization technique established for executing the VM migration in this work is called Adam white shark optimization-based VM placement (AWSO-VMP), produced by merging the Adam optimizer with white shark optimization (WSO). Therefore, load, power consumption, and migration costs have been used to judge the effectiveness of AWSO-VMP; the relevant metrics have been reached at values of 0.133, 0.225 W, and 0.116.

In data mining, sentiment analysis is currently the most popular and active study topic. These days, several social media platforms have been created, with Twitter being one of the most important for exchanging and gathering people's thoughts, feelings, ideas, and attitudes toward specific entities. Because of this, sentiment analysis has become an exciting procedure within the domain of natural language processing (NLP). While various methods for sentiment analysis have been established, there are still perspectives that need to be improved in terms of accuracy and system effectiveness. The suggested architecture develops a deep learning-based sentiment analysis and an optimization-based feature selection to meet it. In [115], the authors analyze the performance of the proposed gated attention recurrent network (GARN) architecture with sentiment 140 dataset. Initially, the preprocessing is used to clean and filter the accessible dataset. The sentiment-based attributes are then decomposed from the pre-processed data using a term weight-based feature-extraction method called the Log

Term Frequency-based Modified Inverse Class Frequency (LTF-MICF) method. In the third phase, a hybrid mutation-based white shark optimizer (HMWSO) is deployed to select features. Recurrent neural networks (RNN) and attention mechanisms are combined in the GARN structure to classify the chosen features into sentiment classifications, such as positive, negative, and neutral. Finally, a comparison test of the suggested and current classifiers' performances is done. Accuracy, precision, recall, and f-measure are evaluated performance metrics with 97.86%, 96.65%, and 96.70% gains, respectively, utilizing the suggested GARN.

The health of people is greatly endangered by hazy weather, so it is crucial to provide precise and trustworthy air pollution concentration forecasts for the benefit of science, locals' way of life, and the environment. Nevertheless, the absence of feature mapping, uncertainty forecast, and outlier detection in current forecasting models leads to forecasts that are not trustworthy. Thus, in [116], It is suggested to use a novel combined deterministic and probabilistic-forecasting system that incorporates several data preparations and combined deterministic and probabilistic forecasts. It is based on a metaheuristic algorithm. Pre-processing and multi-view analysis are performed on the raw data to remove the unpredictable parts. The three deep learning models are combined using a multi-objective white shark optimizer to improve the prediction stability and accuracy. Moreover, a unique interval pseudocode aims to maximize the prediction intervals through development and enhance the reliability of probabilistic forecasting. The Pinball loss function produces prediction intervals at varying confidence levels. According to the simulations, in contrast to the models that were evaluated, the created approach improves forecasting accuracy by up to 64.9001% and 68.4637% at significance levels of 0.05 and 0.1 in the interval sharpness and by a maximum of 60.6772%, 54.9793%, and 38.6876% in the three steps of PM2.5 prediction.

An autonomous wireless temporary network known as a network of mobile ad hoc (MANET) is created using a collection of transportable nodes, such as laptops, smartphones, iPods, etc., that are suitable for the context wherein the networks' infrastructures are dynamic. The most frequent issues that MANETs deal with include limited network longevity, high energy consumption, high traffic overhead, and energy efficiency, all of which affect the network's general architectural design. Thus, an energy-efficient CH option must be provided to address such concerns. Therefore, this [117] suggests a unique model that uses a routing method in an MANET to increase network longevity and energy efficiency. A unique Fuzzy Marine White Shark optimization (FMWSO) algorithm is proposed to find an optimal CH. This approach is achieved by integrating fuzzy operation with two optimization techniques: the White Shark Optimizer and the Marine Predator algorithm. The suggested method consists of three phases: data generation, cluster generation, and CH selection. A unique FMWSO method is proposed to decide the CH selection in an MANET, which improves the network lifetime, topology, and energy consumption while decreasing the overhead rate. To ascertain the system's efficacy, the performance of the suggested FMWSO methodology is contrasted with several other current methods. The minimal energy consumption of the suggested FMWSO strategy is 0.62 MJ, which is less than the previous alternatives.

In recent years, the optimal control problem has been increasingly important in tackling real-world issues. The metaheuristic algorithms have demonstrated their efficacy in solving these problems efficiently and effectively. However, according to the no-free lunch theory, these algorithms might not be able to solve every optimization problem. Consequently, there is always room for the creation of new metaheuristic algorithms [118] suggests the Tyrannosaurus (T-Rex) optimization algorithm (TROA), a novel hunting-based optimization technique. The way that T-Rexes hunt served as inspiration for this algorithm. Twelve benchmark problems and four real-world optimum control issues were used to test this technique. Crow Search Algorithm (CSA), Particle Swarm Optimization (PSO), White Shark Optimizer (WSO), The Differential Evolution (DE) Algorithm, Grey Wolf Optimizer (GWO), Jellyfish Search (JS), and Golden Eagle Optimization (GEO) are the seven well-known optimization techniques with which the performance of the TROA is compared. Compared to these methods, the suggested method's results have shown improvement.

Among the most significant fields of study in photovoltaic (PV) system modeling and design is parameter characterization in PV cell/module models. Diode-based models are commonly used; the most important models are the single-diode model (SDM), double-diode model (DDM), and three-diode model. Therefore, solving the parameter characterization of such models with an objective function can minimize the discrepancy between the estimated and measured current. Recently, metaheuristic optimization techniques have been used to overcome the challenge of rapidly obtaining highly trustworthy and accurate results. Consequently, the basic SDM and DDM are altered in this study, and an objective function is examined using the updated models. Furthermore, an enhanced version of this unique metaheuristic algorithm is suggested by adjusting the force control settings of the White Shark Optimizer (WSO) and adding a chaotic generator to improve WSO's exploitation capability. The PV parameters are extracted using the modified algorithm, which goes by the acronym IWSO. Lakshmanan et al. [74], which compares the modified PV model with the conventional model using the new objective function. The experiment's results proved that IWSO is superior to rival algorithms. With an average score of 1.171 on Freidman's ranking test, the suggested IWSO outperforms all

the chosen algorithms. Modified SDM and DDM have an average accuracy that is 12% higher than that of classic PV models. The results show that IWSO has the best-calculated parameter values, with the slightest variation between the estimated and experimental current.

Breast cancer (BC) is considered the most common malignancy among women worldwide. The disease's survival rate is partly raised by earlier research on BC. Finding malignant areas in the microscopic image of breast tissue is a laborious step in the complex process of diagnosing breast cancer on histopathology images (HIS). There are three ways to find BC on HSI: machine learning (ML), deep learning (DL) based methods, and traditional image processing techniques. The main issues with BC diagnosis on HSI are the ger picture sizes and the significant degree of heterogeneity in the appearance of tumorous regions. With this inspiration, this [119] creates a white shark optimizer with attention-based deep learning for the breast cancer classification (WSO-ABDLBCC) model to provide a computer-aided diagnosis. Using DL methods, the proposed WSO-ABDLBCC methodology accurately classifies breast cancer. To enhance the image quality in the WSO-ABDLBCC technique, guided filtering (GF)-based noise removal is used. Next, the feature vector generation uses the Faster SqueezeNet model with WSO-based hyperparameter optimization. Finally, attention-based bidirectional long short-term memory is used to classify histopathology pictures (ABiLSTM). The WSO-ABDLBCC is well validated experimentally using the benchmark Breakhis database. The accuracy of the suggested model was 95.2%. The testing results showed that when compared to other models already in use, the WSO-ABDLBCC approach achieves better performance.

The primary source of customer opinions about services or products to buy is online reviews. Generally, spam reviews are made to disparage or promote specific targeted goods or services to become well known or profitable. Review spamming is the term for this activity. In recent years, several methods have been suggested to address the issue of spam reviews. Previous research on spam detection has emphasized English reviews more than other languages. Despite the volume of data generated, spam review detection in Arabic online sources is a novel study area. Thus, this [120] creates an automated spam review detection system on Arabic opinion text by utilizing the best Stacked Gated Recurrent Unit (SRD-OSGRU). The primary goal of the SRD-OSGRU model that is being given is to divide Arabic reviews into two categories: spam and genuine. The SRD-OSGRU model that is being described first goes through several stages of data preprocessing to transform the review data into a compatible format. The feature extractors for Bigram and Unigram are then used. This study uses the SGRU model to detect and categorize Arabic spam reviews. The SGRU model's detection efficiency is increased using a white shark optimizer (WSO), because it is laborious to tune hyperparameters through trial and error. Two datasets, especially the DOSC dataset, are used to evaluate the experimental validation of the SRD-OSGRU model. A thorough comparative analysis demonstrated the SRD-OSGRU model's superior performance compared to other contemporary methods.

In a rapidly developing power network, greater voltage levels at long transmission lines are recommended for an efficient power network. The stabilizers that reduce power oscillations are under higher stress due to these elevated voltage levels. Using a white shark optimizer (WSO), this [121] suggests a novel method for the power system stabilizer (PSS) optimal parameter section. On a benchmark test power system, the evaluated optimizer's performance is contrasted with the recently suggested hybrid algorithm from the literature. The generated oscillation damping performance has been examined using the time-domain system parameter specifications. Promising results were obtained with faster setting time characteristics from the suggested WSO-based PSS.

This [122] proposes an enhanced approach-based grid flexibility interpretation for the combined heat and power (CHP) systems with variable renewable energy systems. The White Shark Optimizer (WSO) and the Pelican Optimization Algorithm (POA) are executed simultaneously in the suggested system. The POA strategy improves the updating behavior of the WSO approach; hence, the term enhanced WSO technique. The suggested method's primary goal is to use CHP systems to give the best possible grid flexibility. The demand for sources is estimated for 2030 by applying the suggested technique. The suggested method examines the flexibility in both upward and downward directions and views the optimization challenge as a cost reduction. Power demand from coal, nuclear, wind, hydropower, and solar sources is examined. The suggested approach examined the system's flexibility based on the seasons. Ultimately, the suggested approach's performance is replicated using the MATLAB/Simulink platform, and its results are contrasted with those of other existing methods.

WSO offers unique benefits for optimization tasks, including adaptability to different issue kinds, resilience, simplicity, speed, and precision of solution finding [123]. Higher solution quality and several benefits are associated with the proposed method [124]; among them, many benefits are a few control parameters, a decreased amount of time devoted adjusting control parameter values, less time spent on convergence to optimal solutions, and more trustworthy search capabilities. The reason behind choosing the WSO technique was its adaptability to complex high-dimensional problems, its resilience and ease of use, and its capacity to ensure precise answers by preventing the trapping of local optima [125]. WSO offers several benefits for solving global optimiza-

tion problems, including its predicted adaptability to various optimization problem types. Many problem types require a higher level of flexibility than WSO can provide, with just a few parameters that need to be adjusted. According to the suggested mathematical model, WSO can be used for a broad spectrum of engineering optimization concerns, especially those with enormous dimensionality. A third benefit is expected to be the robustness and simplicity of WSO, which enables fast and precise global solution finding with high convergence speed for challenging optimization problems. Being a strong contender with a broad interest in creating affordable and practical solutions to complex real-world optimization challenges is WSO's fourth advantage for global optimization [65]. White shark optimizer (WSO), a revolutionary bio-inspired metaheuristic algorithm, has drawn interest for its ability to solve global optimization problems [126]. White Shark Optimizer (WSO) was utilized in [127] to optimize the multiple embedding strength (MES) values. Since WSO provides an ideal solution with high precision, the authors have chosen to employ it. According to sum [128], there are several benefits of employing WSO in conjunction with an ensemble classifier for attack detection, including Best Model Selection: Using WSO, the most vital individual classifiers for the ensemble can be chosen depending on how well they perform on the training set, strengthening the ensemble as a whole. Fast Convergence: Compared to other optimization techniques, WSO can help the ensemble classifier converge more quickly to the ideal solution, reducing the time needed to process attacks. Increased Robustness: WSO plus an ensemble classifier helps a model soon converge to the best solution while the ensemble can mitigate the effects of outliers. This increases the model's robustness against inconsistent or noisy data. Improved Handling of Unbalanced Data: WSO can help choose the most effective individual classifiers to handle imbalanced data if one class is underrepresented, improving the ensemble classifier's performance in attack detection. Enhanced Model Performance: The model's overall performance can be improved, leading to more precise and dependable attack detection, by employing WSO to optimize the individual classifiers and ensemble parameters.

The binary character of the feature selection task has been addressed by improving the WSO algorithm in [107]. Initially, two transfer functions translate the continuous domain into binary. To create a high-variability initial population, the modified K-means approach is recommended. Different crossover operators are used to improve the binary-WSO's evolutionary process. BIWSO1 employs transfer functions, BIWSO2 utilizes modified k-means, and BIWSO3 incorporates crossover operators as sophisticated versions. The suggested BIWSO iterations are evaluated using 12 publicly available IDS and IoT datasets.

BIWSO3 is a technology designed to improve Intrusion Detection Systems (IDSs) by improving attack detection performance. The work entails incorporating feature selection strategies into ML-based IDS prediction models to improve them. The White Shark Optimizer (WSO) algorithm, which is optimized for binary feature selection in Intrusion Detection System (IDS) applications, is improved upon in this paper. There are three iterations of the improved algorithm, BIWSO: In BIWSO1, transfer functions are integrated to transform continuous domains into binary spaces; in BIWSO2, a modified K-means algorithm is introduced to produce a more diverse initial population; and in BIWSO3, the algorithm is further refined by adding multiple crossover operators to accelerate the evolutionary process [129].

The [130] presents CGAN-IWSO-ResNet50, a unique technique for phishing attack detection. An enhanced edition of the conditional GAN is employed in the first step to equalize the URL samples. The second phase involves employing TF-IDF and hand-crafted techniques to perform the feature-extraction procedure. To enhance the WSO algorithm's feature selection performance, the WOA algorithm is applied during the feature selection stage. The selected features are applied to the dataset, wherein RGB images represent instances of phishing and legal cases. In the last step, RGB images guide the ResNet50 architecture. In the PhishTank dataset, the accuracy, sensitivity, and precision of the recommended technique are, respectively, 99.65%, 99.12%, and 99.46%.

4 Proposed Model

The idea behind the proposed technique is to combine an ensemble learning model (ELM) with a metaheuristic optimizer to effectively determine the optimal number of weak learners (K-nearest neighbors, Gaussian Naive Bayes, Decision Tree, Gradient Boosting, Random Forest, Adaboost classifier, Extra Trees, Ridge Classifier, Logistic Regression, Multilayer Perceptron (MLP), Quadratic Discriminant Analysis, Bagging, Hist Gradient Boosting, Support Vector Classifier (SVC)) with the highest accuracy and the lowest computational time, ensuring superior performance in classification tasks. The employed ensemble learning model comprises several weak learners and uses the challenging voting technique to calculate the classification accuracy. AUC-ROC is used as the primary performance measure. The binary version of White Shark Optimizer (WSO) is also used to improve the ensemble learning model.

A binary vector is randomly generated, with a length equal to the number of weak learners employed in the ensemble learning model. The target of this vector is to turn on or off each weak learner, where a value of 1 activates the weak learner, and 0 deactivates it. Consequently, a set of

Algorithm 1: WSO algorithm pseudocode

```
Initialize random positions of white sharks x_i, where i = 1, 2, ..., n
Initialize velocity (v_i) of white sharks x_i and parameters of WSO
Evaluate the value of the cost function for each white shark x_i
while (t < T) do
    Update parameters of WSO using predefined equations
    for i = 1 to n do
        Update velocity v_i using predefined equation
        for i = 1 to n do do
            if rand < mv then then
                Update position x_i using predefined equation
            end if
            Update position x_i using predefined equation
            for i = 1 to n do
                if rand < s then
                    Compute distance between white shark and prey
                    if i == 1 then then
                        Update position of white shark w.r.t
                    else
                        Update position of white sharks concerning the head and the prey using a predefined equation
                        Assess and update new positions
                    end if
                end if
            end for
        end for
    end for
    t = t + 1
end while
Return the optimal solution x_best and the corresponding objective function value f_best
```

weak learners is activated during each training iteration. It contributes to the ensemble learning model classification process, while the remaining weak learners are deactivated and excluded from the classification process.

Therefore, this binary vector is injected into the ensemble learning model; thus, specific weak learners are activated or deactivated depending on the presence of 1 s or 0 s in the vector. Subsequently, the ensemble learning model performs its standard process of bootstrap aggregation classification, utilizing only the active weak learners and the input training data. Finally, it calculates the final classification outcome using the complex voting method shown in Fig. 7.

The final classification performance metric is fed back to the binary WSO to start its first optimization iteration. The resulting outcome from the WSO is a new binary vector, which is once again injected into the ensemble learning model for the subsequent training phase, following the methodology outlined in the preceding paragraph, and its classification performance is assessed anew. This iterative training process is repeated until a satisfactory level of performance is reached, with the highest classification accuracy achievable and minimal computational time. It is worth mentioning that while computational time is a significant consideration, the authors prioritize the accuracy objective more than the computational time objective.

The outcome of this hybridization between the ensemble learning model and the metaheuristic optimizer is an adaptive ensemble learning model comprised of a select few weak learners yet achieving high classification performance with minimal processing time. A customized ensemble learning model will feature its unique weak learners for each classification issue. The developed hybridization process is illustrated in Fig. 8.

It is possible to discuss the process of the proposed model by utilizing the Algorithm 2.

5 Experiment and Results

This section will provide the techniques used to normalize the datasets as long as the process is used to avoid overfitting. A description of the datasets used and a presentation of the results obtained. In addition, there are three subsections for the weak learners identified for each dataset, the execution

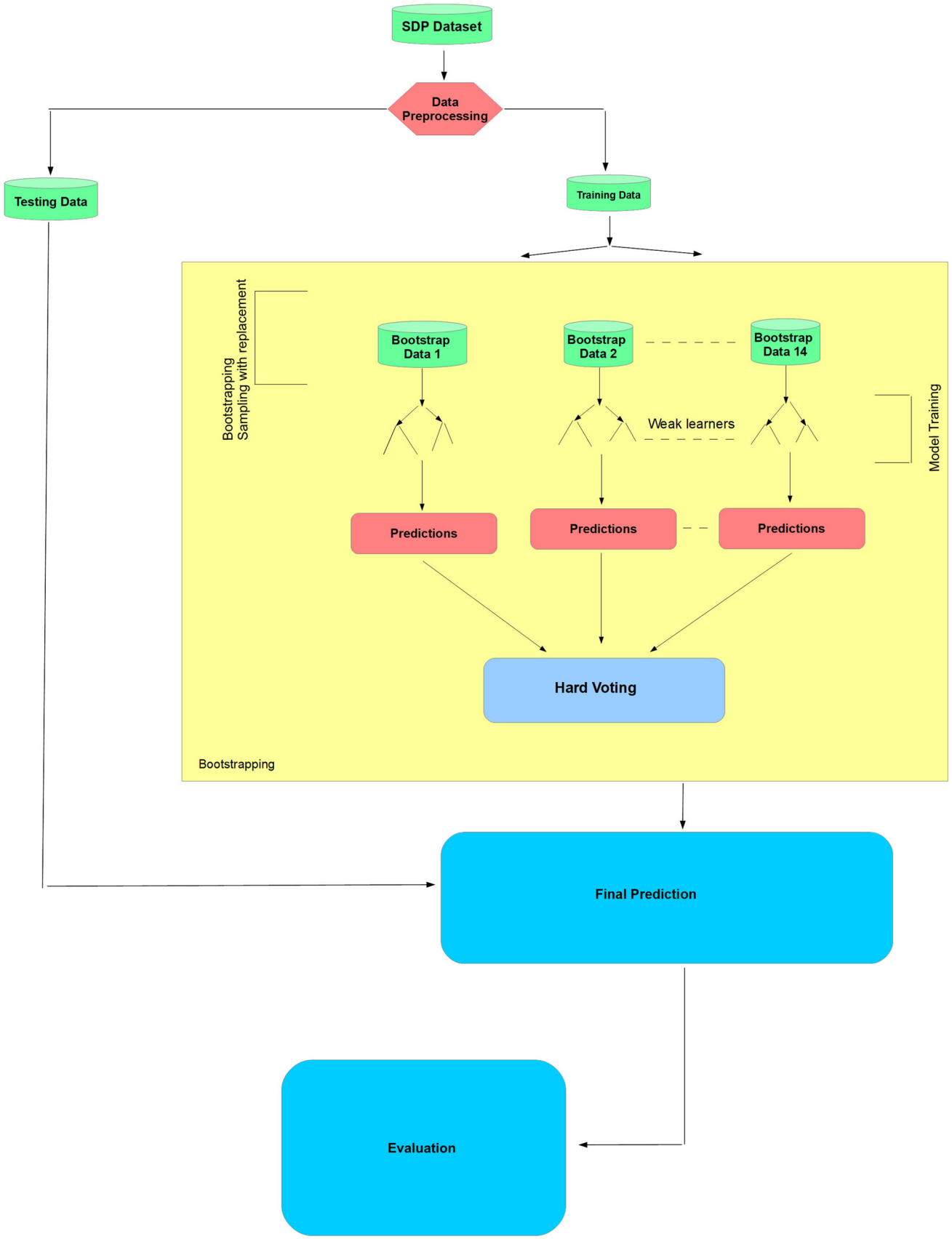

Fig. 7 Detailed ensemble process

Algorithm 2: Proposed model pseudocode

```
Initialize:
        num_weak_learners = 14                                   ▷ number of weak learners
        max_iterations = 30                                      ▷ Maximum number of WSO iterations
        max_ensemble_iterations = 10                             ▷ Maximum number of ensemble iterations
        binary_vector = GenerateRandomBinaryVector(num_weak_learners)   ▷ Initial random binary vector
function GenerateRandomBinaryVector(n)
    binary_vector = []
    for i ← 1 to n do
        Append a random 0 or 1 to binary_vector
    end for
    return binary_vector
end function
function TrainEnsemble(binary_vector, training_data)
    active_learners = []
    for i ← 1 to num_weak_learners do
        if binary_vector[i] == 1 then
            Add weak_learners[i] to active_learners
        end if
    end for
    predictions = []
    for each learner in active_learners do
        prediction = learner.Predict(training_data)
        Append prediction to predictions
    end for
    final_prediction = HardVoting(predictions)
    accuracy = CalculateAccuracy(final_prediction, true_labels)
    auc = CalculateAUC(final_prediction, true_labels)            ▷ New: Calculate AUC
    return accuracy, auc
end function
function HardVoting(predictions)
    final_votes = []
    for i ← 1 to length(predictions[0]) do
        votes = Get votes for data point i
        majority_vote = mode(votes)
        Append majority_vote to final_votes
    end for
    return final_votes
end function
while wso_iterations < max_iterations do
    ensemble_accuracy_sum = 0                                    ▷ Sum of accuracies across ensemble iterations
    ensemble_auc_sum = 0                                         ▷ Sum of AUCs across ensemble iterations
    for ensemble_iteration ← 1 to max_ensemble_iterations do     ▷ Ensemble internal loop
        accuracy, auc = TrainEnsemble(binary_vector, training_data)
        ensemble_accuracy_sum+ = accuracy
        ensemble_auc_sum+ = auc
    end for
    average_ensemble_accuracy = ensemble_accuracy_sum/max_ensemble_iterations   ▷ Average accuracy over ensemble iterations
    average_ensemble_auc = ensemble_auc_sum/max_ensemble_iterations   ▷ Average AUC over ensemble iterations
    if average_ensemble_accuracy > best_accuracy then
        best_accuracy = average_ensemble_accuracy
        best_auc = average_ensemble_auc                          ▷ Track the best AUC
        best_vector = binary_vector
    end if
    if best_accuracy == 1.0 then
        Break
    end if
    binary_vector = WSO_optimize(binary_vector, average_ensemble_accuracy)
    wso_iterations+ = 1
end while
Output "Best Binary Vector:," best_vector
Output "Best Accuracy Achieved:", best_accuracy
Output "Best AUC Achieved:", best_auc                            ▷ Output the best AUC as well
final_accuracy, final_auc = TrainEnsemble(best_vector, test_data)
Output "Final accuracy on test data:" final_accuracy
Output "Final AUC on test data:, "final_auc                      ▷ Final AUC on test data
```

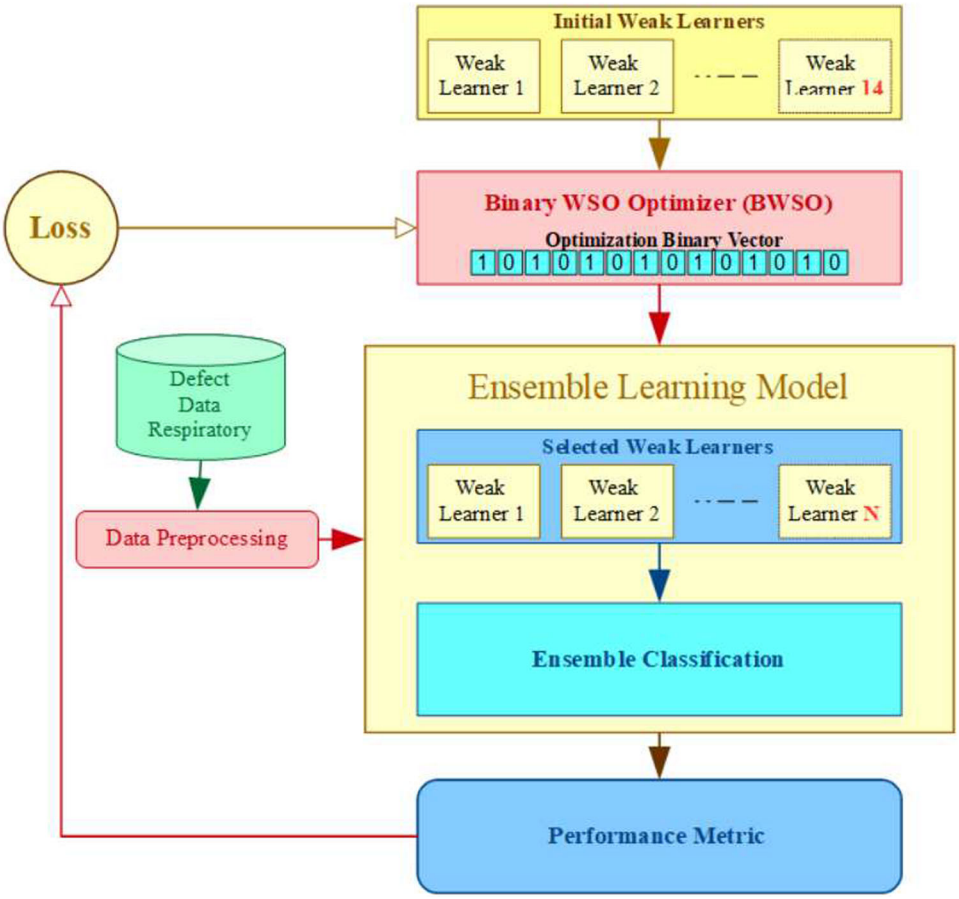

Fig. 8 Proposed technique

time of the experiment on each dataset investigated, and the statistical analysis of the results obtained, respectively.

5.1 Experimental Setup

The proposed techniques were developed using Python 3.10 in Colab. In addition, a Windows 11 Gen 64-bit installed on a Workstation with an Intel(R) Core(TM) i5-1135G7 CPU @ 2.4 GHz processor and 8.0 GB RAM was used.

5.2 Dataset

For experimental analysis, the work uses a large collection of fifteen benchmark datasets drawn from three well-known open-source repositories: NASA, SOFTLAB, and ReLink. **NASA Repository**: The dataset from this repository contains software metrics at the method level that were taken from eight different NASA software projects written in Java, C, and C++. Even if different projects have different sets of available software metrics, certain projects have different metrics. Examples of software measurements include 38 for PC5,36 for PC2, and 37 for five projects (MW1, PC1, PC3, PC4, and CM1). Furthermore, MC2 contains 39 metrics. These datasets come from the MDP and PROMISE repositories; MDP [131] is where the data for the MDP dataset is sourced. **SOFTLAB repository**: five datasets from this repository are included, such as AR1, AR3, AR4, AR5, and AR6 [132]. **ReLink Repository**: two separate projects, such as Zxing, and Apache, make up the ReLink dataset. A detailed description of these datasets is provided in Table 2.

5.2.1 Data Pre-processing

The utilized datasets were randomly divided into two segments: 70% allocated for training and the remaining 30% designated for testing. Furthermore, to standardize the scale of all data, all input values are normalized to the range [0, 1]. All tests' input data were linearly transformed via the Min–Max normalization approach. As demonstrated in Eq. 20, Min–Max normalization is employed to scale the data

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}. \tag{20}$$

In Eq. 20, x' represents the normalized value of x, scaled between 0 and 1 based on the minimum and maximum values of the dataset. In addition, the authors adopted an early stopping technique for relatively small-size datasets. Because of

Table 2 Datasets description

Project group	Datasets	Refs.	Count of instances	Defective	Not-defective	Count of software metrics	% defects
NASA	CM1	[133–135]	327	42	285	37	12.8
	MW1	[131]	253	27	226	8	10.6
	MC2	[131, 135]	126	45	81	39	35.7
	PC1	[133–135]	706	61	645	37	8.6
	PC2	[131, 135]	746	16	729	36	2.1
	PC3	[131, 135]	1078	135	943	37	12.5
	PC4	[131, 135]	1288	178	1110	37	13.8
	PC5	[131, 135]	1712	472	1240	38	27.5
Relink	Apache	[136]	194	98	96	26	50
	Zxing	[136]	399	118	281	41	29.5
SOFTLAB	ar1	[132, 135]	121	9	112	29	7.4
	ar3	[135]	63	8	55	14	12.6
	ar4	[135]	107	20	87	23	18.6
	ar5	[135]	36	8	28	29	22.2
	ar6	[135]	101	15	86	17	14.8

this, overfitting is avoided using this technique by running the number of experimental iterations to 21.

5.2.2 Dataset Analysis

In this section, an analysis of the data used in this research was done. First, each class's data distribution was examined for each class separately to determine whether the distribution was normal, close, or abnormal (see, Fig. 9). Then, analyses were done that included the Cohen kappa coefficient and Matthews correlation coefficients for the data that contained an abnormal distribution, and to solve this problem, this study employed SMOTE to augment the instances in the minority class or to lessen the cases in the majority class in the context of oversampling or undersampling, respectively [137].

Concerning the Influence of Feature Variance on Classification Accuracy, we acknowledge that datasets such as MW1, which comprise merely eight metrics, in contrast to datasets like ar3 (14 metrics) and ar6 (17 metrics), may exhibit a disparity in feature representation. A reduced number of features may constrain the depth of the input data, thus impairing the model's capacity to identify intricate patterns. Conversely, an increase in features does not necessarily enhance performance, as it may contribute noise or extraneous information.

Table 3 compares numerous projects using four essential metrics: True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP), as well as the computed values for Matthews Correlation Coefficient (MCC) and Cohen's Kappa. The projects exhibit considerable variability in performance, with PC4 achieving the greatest MCC (0.4205) and Cohen's Kappa (0.3376), signifying a more favorable equilibrium between positive and negative predictions relative to other projects. Conversely, projects such as CM1 exhibit negative values for both criteria, indicating subpar predictive performance. Furthermore, other projects, such as Zxing and Ar6, demonstrate poor MCC and Kappa values, signifying a deficiency in concordance between predicted and actual classifications. The prevailing tendency indicates that whereas several programs attain considerable predictive accuracy, others falter, underscoring the necessity for additional investigation and maybe enhanced approaches. Projects such as PC5 and PC1 demonstrate favorable MCC values exceeding 0.3, signifying a substantial correlation between expected and actual results, suggesting that they could gain from focused improvements to enhance their classification efficacy.

5.3 Weak learners

This part will examine the prior applications of the algorithms utilized in this study, along with their significance and advantages, as highlighted by earlier research. Table 4 illustrates the weak learners utilized in this study as documented in prior research on software defect classification.

5.4 Results

The experiment comparing the effectiveness of an optimized ensemble learning model (OM) utilizing the binary White Shark Optimizer (WSO) metaheuristic algorithm with that of a standard ensemble learning model (EM) is shown in Table 5. With 15 distinct benchmark datasets, the experiment seeks to find an optimal number of weak learners to

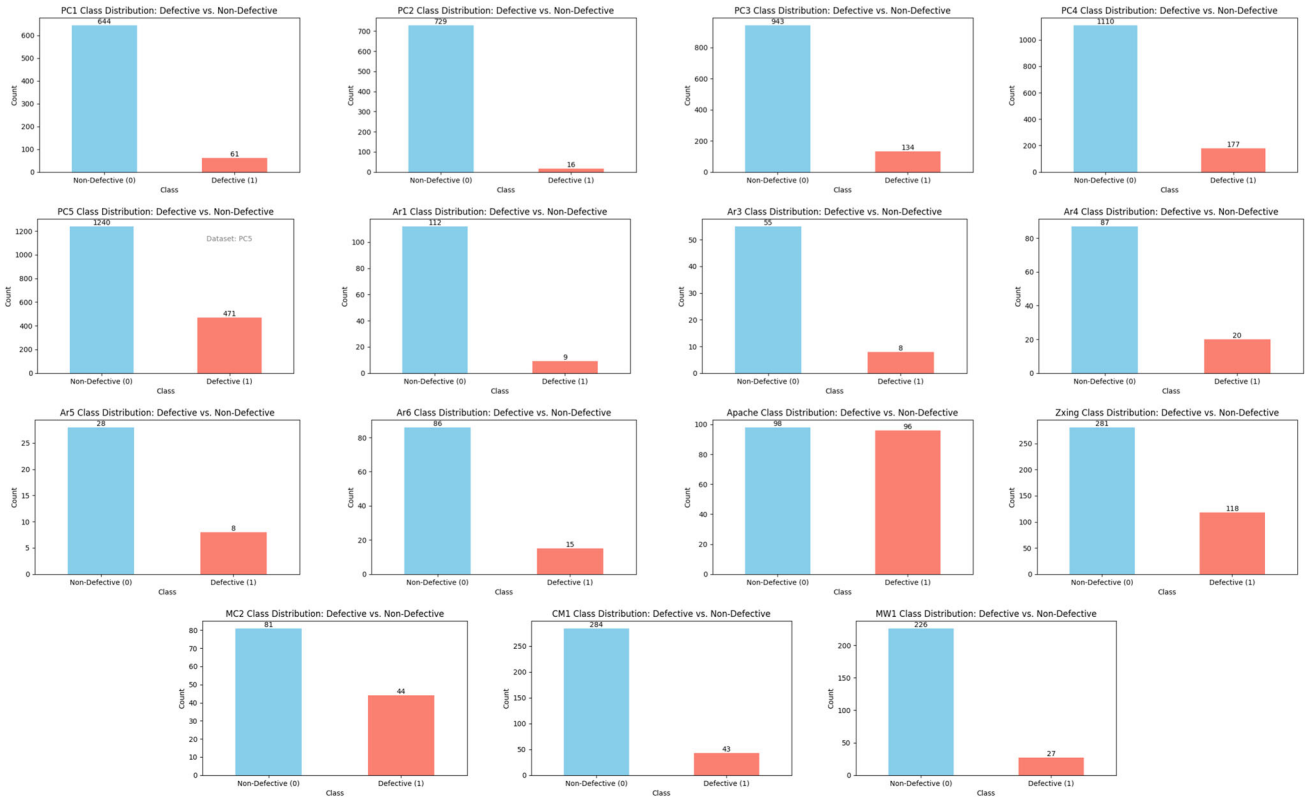

Fig. 9 Data distribution per classes

Table 3 Dataset comparative analysis

Project	TN	FP	FN	TP	MCC	Cohen Kappa
Zxing	71	12	30	7	0.0564	0.0516
PC5	331	36	93	54	0.3201	0.3047
PC4	335	2	38	12	0.4205	0.3376
PC3	274	5	38	7	0.2520	0.1988
PC2	216	0	8	0	0	0
PC1	193	4	11	4	0.3315	0.3141
MW1	62	0	13	1	0.2430	0.1115
MC2	21	9	2	6	0.3753	0.3407
CM1	82	3	14	0	−0.0717	−0.0525
Ar6	24	0	7	0	0	0
Ar5	8	0	2	1	0.5164	0.4211
Ar4	25	1	5	2	0.3516	0.3125
Ar3	17	0	0	2	1	1
Ar1	35	0	2	0	0	0
Apache	24	8	10	17	0.3827	0.3818

attain maximum accuracy as measured by several metrics. The following discusses these findings:

Regarding AUC-ROC, Accuracy, Precision, Recall, F1-score, and Specificity performance metrics, the optimized OM model consistently performs better than the EM model across all datasets. All classification metrics values for OM are regularly better than those of EM at the minimum, maximum, and average levels, suggesting that the metaheuristic optimization procedure improves the model's classification performance.

Generally speaking, OM's standard deviation values are lower (better) than EM's, indicating that the WSO optimization algorithm lessens the variability in model performance between experiment iterations. This suggests that the optimized OM model produces more consistent and dependable outcomes, which is advantageous in practical applications where stable performance is essential.

Depending on the dataset, OM outperforms EM in terms of performance, with some datasets seeing more considerable gains than others. Significant increases in AUC-ROC accuracy are seen with OM for datasets, such as Ar1, Ar3, Ar4, Ar5, and PC4, suggesting that the optimization method works exceptionally well for these issues.

In addition, across various datasets, the OM typically shows better precision, recall, and F1-score values than the EM. More recall suggests fewer false negatives; more accuracy shows fewer false positives, and a higher F1 score represents a better trade-off between recall and precision. These enhancements imply that the OM model can achieve

Table 4 Weak learners usage in the field of software defect classification

Weak learner	Advantages	Refs.
SVM	Faster and better with small datasets. Small-sample, nonlinear, high-dimensional problem solver. Has kernel functions and quadratic programming	[26, 36, 77, 133, 138, 139]
RF	Little tuning, great generalization, and minimal training time despite dataset size. Many fields use RF due to its bagging and random subspace technique	[48, 140–143]
DT	It manages missing data, nonlinear relationships, and continuous and categorical variables and is computationally efficient and easy to use	[32, 133, 144–146]
Knn	Ability to process numerical and categorical data. Manages defective-non-defective class imbalance	[147–152]
Gradient Boost	Can even improve computing time complexity and performance. Fast and effective	[101, 153–156]
AdaBoost	Focuses on misclassified data, manages high-dimensional data, accommodates noisy data, estimates feature relevance, and overfits less than a decision tree or random forest	[135, 144, 148, 157, 158]
Extra Trees	More variation is reduced by full cut-point, attribute randomization, and ensemble averaging than weaker randomization strategies	[152, 159–162]
Ridge	Simplicity, lower computing demands, and better interpretability than other algorithms	[163–165]
Logistic Regression	In cross-project defect prediction research, the most commonly utilized modeling technique is followed by Naïve Bayes due to its superior performance. Variables can be discrete or mixed and not normally distributed	[33, 146, 152, 166, 167]
MLP	An MLP network's bias and weight parameters are adjusted to maximize data fit during training. The network processes input and changes parameters using learning techniques iteratively to improve performance during training	[168–172]
Quadratic Discriminant Analysis (QDA)	QDA can accurately model more problems than linear approaches, since it assumes a quadratic decision boundary. QDA reduces misclassification by permitting distinct covariance matrices for each group	[173–177]
Bagging	Handle imbalanced datasets. Bagging can uniformly sample datasets and perform parallel computing. Hence, learners using the bagging framework have greater generalization and processing speed	[142, 178–182]
Hist Gradient Boosting (HGB)	Employs feature histograms to quickly and accurately choose the best splits. HGB uses less memory and processes faster than GBM	[178, 183–185]
Naïve Bayes	No tuning parameter. It predicts class labels using Bayes theory	[186–189]

Table 5 Results obtained by the basic ensemble learning model and the optimized model

Dataset	Model	AUC				Accuracy	Precision	Recall	F1 score	Specificity
		Worst	Best	Avg	Std	Avg	Avg	Avg	Avg	Avg
Apache	EM	0.5695	0.6939	0.6376	3.73E−02	0.6429	0.5902	0.5966	0.5868	0.6786
	OM	0.6834	0.8326	0.7651	2.98E−03	0.7714	0.7083	0.7160	0.7041	0.8143
Ar1	EM	0.4783	0.7500	0.5116	6.93E−02	0.9114	0.0714	0.0357	0.0476	0.9876
	OM	0.5643	0.8850	0.6037	1.17E-16	0.7881	0.0843	0.0421	0.0562	0.8970
Ar3	EM	0.5000	0.8274	0.7679	1.54E−01	0.9286	0.8571	0.5357	0.6429	0.7864
	OM	0.5455	0.9027	0.8377	3.01E−04	0.8975	0.9351	0.5845	0.7014	0.7896
Ar4	EM	0.5000	0.7706	0.6248	8.35E−02	0.8019	0.6155	0.3000	0.3799	0.7983
	OM	0.5435	0.8376	0.6791	7.26E−03	0.8717	0.6690	0.3261	0.4130	0.8678
Ar5	EM	0.5000	0.8347	0.6667	1.05E−01	0.7321	0.7143	0.4048	0.4929	0.8286
	OM	0.5600	0.9349	0.7467	8.04E−04	0.8200	0.8000	0.4533	0.5520	0.9280
Ar6	EM	0.4667	0.6667	0.6024	8.52E−02	0.7687	0.6190	0.2143	0.3175	0.7705
	OM	0.5273	0.7533	0.6807	1.76E−02	0.8686	0.6995	0.2636	0.3905	0.8706
CM1	EM	0.4912	0.6784	0.5495	6.51E−02	0.6255	0.2957	0.1190	0.1532	0.8199
	OM	0.6243	0.8621	0.6984	1.02E−03	0.7950	0.3758	0.1513	0.1947	0.8937
MC2	EM	0.4821	0.7312	0.6909	1.08E−01	0.7600	0.3737	0.5893	0.4477	0.7925
	OM	0.6297	0.9549	0.9023	9.80E−04	0.9926	0.4880	0.7696	0.5847	0.7973
MW1	EM	0.5000	0.6646	0.5512	4.88E−02	0.8081	0.4464	0.1286	0.1819	0.8739
	OM	0.5445	0.7238	0.6003	2.00E−04	0.8800	0.4862	0.1400	0.1981	0.9516
PC1	EM	0.5339	0.8062	0.6654	6.75E−02	0.9063	0.5238	0.3750	0.4007	0.7557
	OM	0.5718	0.8634	0.7126	7.00E−05	0.9706	0.5610	0.4016	0.4291	0.8094
PC2	EM	0.4862	0.6250	0.5231	5.01E−02	0.6740	0.6151	0.5357	0.6337	0.7526
	OM	0.6330	0.8138	0.6811	1.32E−02	0.8776	0.8008	0.6975	0.8251	0.8294
PC3	EM	0.5000	0.7508	0.5812	6.70E−02	0.8168	0.4717	0.2327	0.2445	0.7553
	OM	0.5489	0.8243	0.6380	1.74E−05	0.8967	0.5178	0.2554	0.2684	0.8292
PC4	EM	0.5961	0.7836	0.6658	5.36E−02	0.6766	0.6861	0.3973	0.4474	0.6043
	OM	0.7159	0.9411	0.7996	7.45E−05	0.8126	0.8240	0.4772	0.5373	0.7258
PC5	EM	0.5473	0.6739	0.6084	4.68E−02	0.7224	0.5827	0.3143	0.3968	0.9025
	OM	0.7728	0.8360	0.8081	2.49E−02	0.8261	0.8406	0.7866	0.8121	0.8406
Zxing	EM	0.5027	0.6775	0.5605	4.51E−02	0.6455	0.5229	0.2512	0.3312	0.8697
	OM	0.6073	0.8184	0.6771	1.35E−02	0.7798	0.6316	0.3035	0.4000	0.9467

more balanced precision and recall performance and better discriminate between positive and negative instances.

Furthermore, the OM model's performance gains varied among datasets. It shows notable improvements in Accuracy, Precision, Recall, F1 Score, and Specificity performance measures for specific datasets (e.g., Ar1, Ar3, and MC2), suggesting that the optimization method is incredibly successful for these datasets. However, the improvements are negligible for datasets such as Ar6 and PC3, indicating that the features of these datasets may affect how well the optimization procedure works.

The percentage of actual negative cases the model accurately detected is specificity. The OM model tends to retain or slightly enhance specificity compared to the EM model, suggesting its ability to correctly identify negative cases, even if specificity values are generally high for both models.

The consistent performance gains observed across several datasets demonstrate the optimization approach's scalability and generalizability. This implies that performance can be maintained while using the optimized OM model with the binary-WSO algorithm for various problems.

The experiment concludes by showing how binary WSO can improve ensemble learning models' performance across a range of performance criteria over various classification issues. The findings demonstrate how optimization strategies can enhance classification precision and dependability in ML applications.

Figure 10 shows that the OM model demonstrates a distinct superiority in essential performance metrics, including accuracy, AUC, and specificity, indicating that it is the more resilient model overall. Its enhanced specificity suggests improved capability in identifying negative instances. The

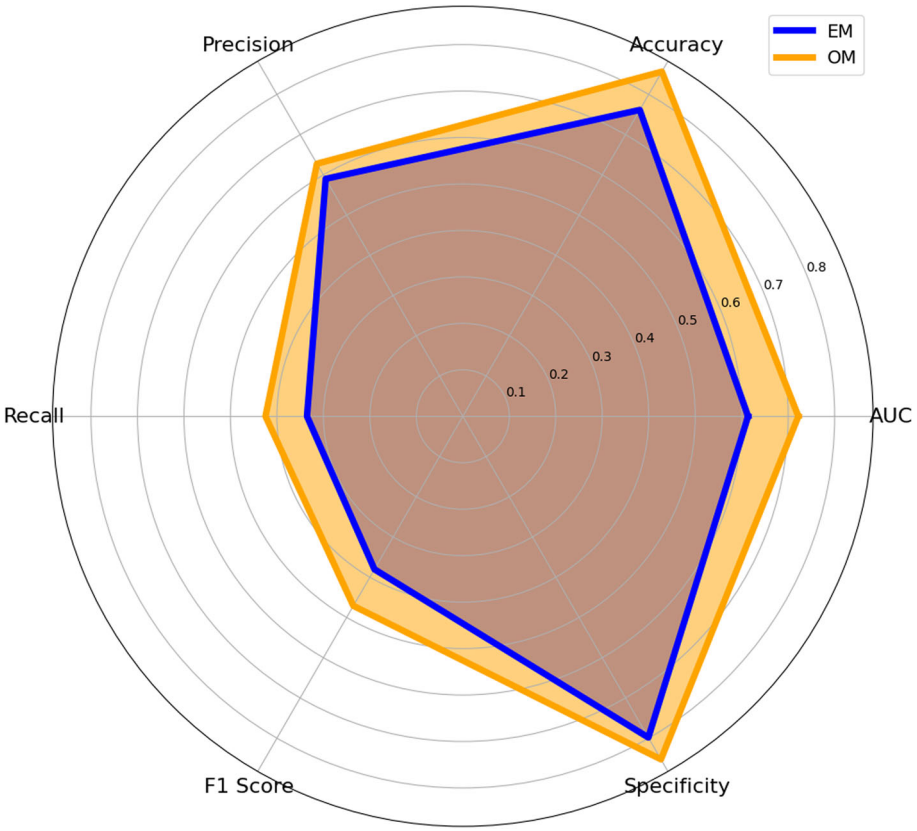

Fig. 10 Average performance metrics

EM model does not exhibit superior performance in any area compared to the OM model; nonetheless, the disparities in recall, precision, and F1 score between the two models are minimal. Consequently, the EM model may remain suitable when the equilibrium between precision and recall is prioritized over total accuracy or specificity maximized.

Table 6 compares the performance of two models, EM and OM, on multiple datasets, using AUC-avg for testing and training. Overall, the OM model outperforms the EM model in testing and training, with higher AUC-avg scores across most datasets. For example, OM gets a test AUC of 0.7651, much higher than EM's 0.6376. Similarly, in the MC2 dataset, OM outperforms EM, with a test AUC of 0.9023 versus 0.6909. This pattern is constant across the vast majority of datasets. When comparing test and training AUC scores, both models exhibit greater AUC values during training than during testing. The difference between test and training AUC is often less for the OM model, implying higher generalization. For example, in the Zxing dataset, OM has a smaller gap (0.6771 in testing vs. 0.714 in training), whereas EM has a greater gap (0.5605 in testing vs. 0.6023 in training). Overall, OM outperforms the other datasets regarding predictive performance and robustness.

The results of the comparison between OM and EM regarding AUC (avg) for training and testing data are visualized in Fig. 11.

5.5 Defined Weak Learners for Each Dataset

After several experimental runs (21 runs), the results show that each dataset has a different number and type of weak learners that are fit. These weak learners are illustrated in Table 7.

The optimal number of weak learners varies considerably among the datasets used. The idea is that there is not a "one-size-fits-all" solution in ensemble learning. The complexity and properties of the data determine the ideal number. In the best setups, specific weak learners regularly show up (e.g., Decision Tree, Random Forest). This implies that these learners could be strong performers on various datasets, especially in classification tasks using the AUC-ROC statistic.

Furthermore, "Ar" datasets benefit from having more weak learners, maybe because of their intrinsic complexity. Conversely, "PC" datasets may be less complex or have better feature representations, because they produce good results with fewer weak learners.

The number of weak learners has not increased but has decreased. Also, the search space has not expanded but has shrunk. Consequently, the optimization process has become faster and more efficient.

Table 6 Average AUC for training and testing data

Dataset	Model	AUC-avg(test)	AUC-avg(training)
Apache	EM	0.6376	0.7351
	OM	0.7651	0.8357
Ar1	EM	0.5116	0.6541
	OM	0.6037	0.7145
Ar3	EM	0.7679	0.8153
	OM	0.8377	0.8905
Ar4	EM	0.6248	0.7027
	OM	0.6791	0.7434
Ar5	EM	0.6667	0.7367
	OM	0.7467	0.8021
Ar6	EM	0.6024	0.6811
	OM	0.6807	0.7554
CM1	EM	0.5495	0.6209
	OM	0.6984	0.7637
MC2	EM	0.6909	0.7529
	OM	0.9023	0.9556
MW1	EM	0.5512	0.6248
	OM	0.6003	0.6816
PC1	EM	0.6654	0.7063
	OM	0.7126	0.7726
PC2	EM	0.5231	0.6062
	OM	0.6811	0.7497
PC3	EM	0.5812	0.6679
	OM	0.638	0.7231
PC4	EM	0.6658	0.7449
	OM	0.7996	0.8588
PC5	EM	0.6084	0.7045
	OM	0.8081	0.8752
Zxing	EM	0.5605	0.6023
	OM	0.6771	0.714

5.6 Comparison with Other Meta-heuristics Approaches

In this section, the results of om are compared to other metaheuristic algorithms, and the following tables show the results:

Table 8 evaluates the performance of different prediction models on the Apache dataset. OM outperforms Cuckoo Search, Firefly, Harris Hawks, and Particle Swarm in all categories. OM had the highest average AUC score of 76.51% and the lowest variation (2.98e−3), indicating robust and reliable prediction. OM outperforms all models in accuracy with an average AUC of 77.14%. Its high precision (70.83%), recall (71.60%), and F1-score (70.41%) help balance false positives and negatives. Cuckoo Search shows competitive results with an average AUC of 70.80% and decent precision (71.28%), but it trails OM in precision and recall. Firefly, Harris Hawks, and Particle Swarm perform well but are less effective than OM. Firefly's average AUC of 69.93% and low precision and recall might be improved. Harris Hawks has the highest specificity (78.18%) but worse precision and recall than OM. Particle Swarm equals OM but not better. OM outperforms AUC, precision, accuracy, and recall on the Apache dataset, making it the best model. This suggests that OM may be effective for predicting tasks that require consistency and strength. The AUC value comparisons are illustrated in Fig. 12.

Table 9 compares metaheuristic algorithm performance using the Ar4 dataset. OM outperforms other approaches with an average AUC of 67.91%. It has good accuracy (87.17%), precision (66.90%), recall (32.61%), F1 score (41.30%), and specificity (86.78%). These indicators show balanced performance, yet the OM model's precision and recall trade-off of positive instances and accuracy. Firefly

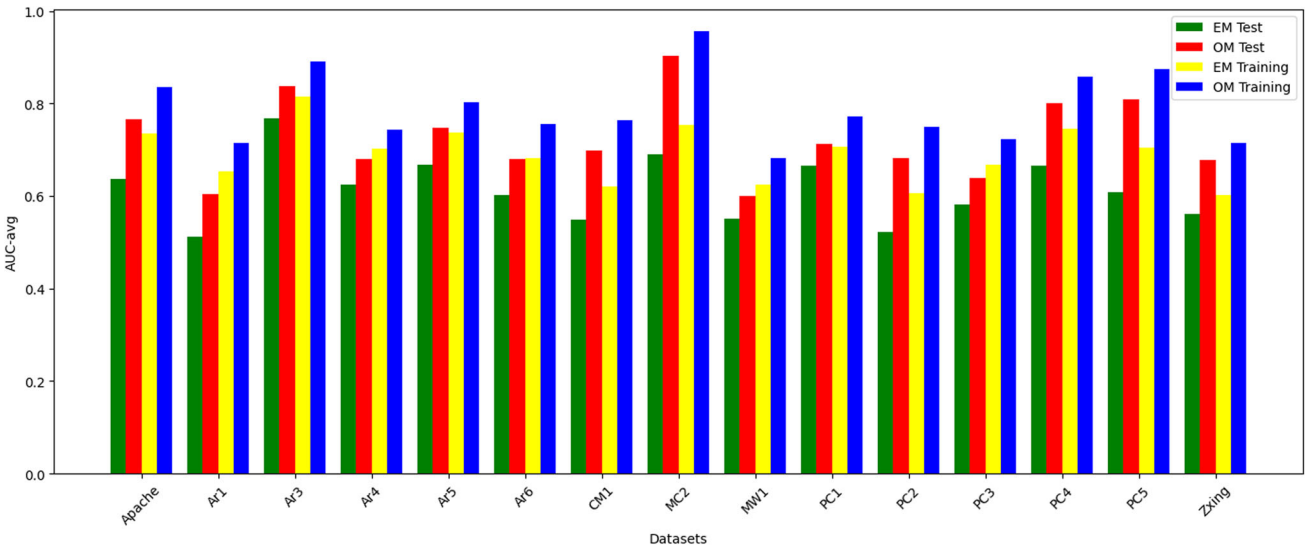

Fig. 11 Average AUC for training and testing data

Table 7 Defined weak learners for each dataset

Dataset	Defined weak learners
Apache	GaussianNB, Decision-Tree, Gradient-Boosting, Random-Forest
Ar1	Gradient-Boosting, Random-Forest, AdaBoost, SVM
Ar3	Gradient-Boosting, Random-Forest, Logistic-Regression, MLP, SVM
Ar4	Decision-Tree, AdaBoost, Extra-Trees, Logistic-Regression, SVM
Ar5	KNeighbors, GaussianNB, Decision-Tree, HistGradient-Boosting
Ar6	Decision-Tree, Random-Forest, Extra-Trees, Logistic-Regression, SVM
CM1	Decision-Tree, Random-Forest, AdaBoost, Extra-Trees, Logistic-Regression
MC2	Decision-Tree, Random-Forest, SVM
MW1	Decision-Tree, AdaBoost, MLP, Bagging, HistGradient-Boosting
PC1	Random-Forest, Extra-Trees, Logistic-Regression, MLP, HistGradient-Boosting
PC2	Random-Forest, AdaBoost, Logistic-Regression
PC3	Decision-Tree, Gradient-Boosting, Random-Forest, MLP, SVM
PC4	Decision-Tree, AdaBoost, HistGradient-Boosting
PC5	Decision-Tree, Random-Forest, AdaBoost, Bagging
Zxing	Random-Forest, Logistic-Regression, Bagging

Table 8 Comparison of WSO with other optimizers on Apache dataset

Dataset	Model	AUC				Accuracy	Precision	Recall	F1 Score	Specificity
		Worst	Best	Avg	Std	Avg	Avg	Avg	Avg	Avg
Apache	OM	0.6834	0.8326	0.7651	2.98e−3	0.7714	0.7083	0.7160	0.7041	0.8143
	Cuckoo search	0.6939	0.7393	0.7080	1.41e−2	0.7128	0.6738	0.6706	0.6701	0.7455
	Firefly	0.6644	0.7166	0.6993	1.65e−2	0.7000	0.6459	0.6941	0.6684	0.7045
	Harris Hawks	0.6578	0.7620	0.7115	3.96e−2	0.7205	0.7021	0.6412	0.6671	0.7818
	Particle Swarm	0.6872	0.7393	0.7057	1.79e−2	0.7103	0.6669	0.6706	0.6685	0.7409

and Cuckoo Search models have similar AUCs of 70.18% and 73.29%. Among these models, Firefly has the best average precision (68%) and recall (56%). Harris Hawks can classify well with an average AUC of 68.06% and excellent specificity (94.12%). The Particle Swarm model has a lower average AUC of 68.18% but good precision (63.50%) and recall (44%). Firefly has an excellent average AUC, high accuracy, and better precision and recall than OM. AUC and specificity are notable in the Cuckoo Search and Harris Hawks models. This comparison reveals that the OM model is reliable and competitive across various measures. Figure 13 illustrates the comparison of AUC values.

Refer to Table 10 for metaheuristic algorithms on the Ar5 dataset. The OM model can be classified because of its high average AUC of 74.67%. Precision (80%), recall (45.33%), and accuracy (82%) are similarly remarkable. Its 92.80% specificity shows that it can dependably identify negative cases, but its recall implies it may struggle to discover positive ones. In terms of average AUC (79%), precision (96.67%), and recall (60%), Cuckoo Search is the best strategy for detecting positive cases. It has low false positives and a good specificity of 98%. A trade-off between precision and overall area under the curve is suggested by Cuckoo Search's lower average AUC than the OM model. Firefly performs well with an average AUC of 77.67% and accuracy of 81.25%. Recall of 63.33% and precision of 85% show its capacity to detect positive cases with a high specificity of 92%. Harris Hawks performs well with a 68.67% average AUC and 75% accuracy. Its precision and recall are high, and its AUC and F1 scores are lower than the best models. Particle Swarm has the lowest average AUC (63.33%), accuracy (75%), and recall (36.67%). Its overall performance is less competitive than the other algorithms despite its high specificity of 90%. Cuckoo Search has the highest precision and recall, but the OM model has good AUC and specificity. Firefly's balanced performance makes it practical. Harris Hawks and Particle Swarm are competitive but weaker in AUC and classification than the leading models. We can observe the AUC values compared in Fig. 14.

Table 11 evaluates metaheuristic algorithms' performance on the CM1 dataset. The OM model is one of the best algorithms is the OM model, with an AUC of 69.84%. This shows its ability to distinguish good from bad cases. OM's accuracy of 79.50% and specificity of 89.37% show that it can efficiently identify negative cases while maintaining balanced performance. Although Cuckoo Search has a lower average

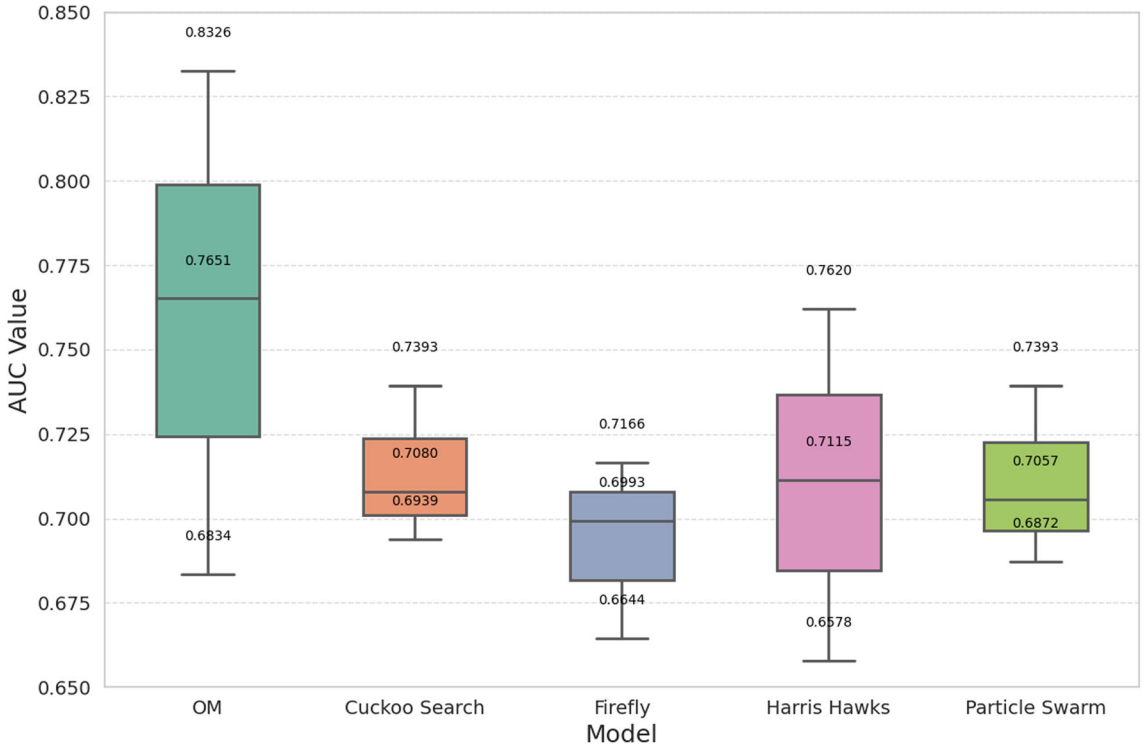

Fig. 12 AUC comparison between WSO and other metaheuristics on Apache dataset

Table 9 Comparison of WSO with other optimizers on Ar4 dataset

Dataset	Model	AUC				Accuracy	Precision	Recall	F1 Score	Specificity
		Worst	Best	Avg	Std	Avg	Avg	Avg	Avg	Avg
Ar4	OM	0.5435	0.8376	0.6791	7.3e−3	0.8717	0.6690	0.3261	0.4130	0.8678
	Cuckoo search	0.6412	0.7706	0.7018	4.56e−2	0.8227	0.6883	0.4800	0.5498	0.9235
	Firefly	0.7000	0.7412	0.7329	1.74e−2	0.8273	0.6800	0.5600	0.5943	0.9059
	Harris Hawks	0.5412	0.7706	0.6806	7.63e−2	0.8227	0.6933	0.4200	0.5112	0.9412
	Particle Swarm	0.6412	0.7706	0.6818	4.15e−2	0.8136	0.6350	0.4400	0.5156	0.9235

AUC of 58.98%, it has the highest accuracy (47.62%) and specificity (96.84%). It is modestly successful at discovering positive instances but good at preventing false positives. However, its recall (21.11%) and F1 Score (27.10%) are worse than OM, indicating that it misses many good cases. Like Cuckoo Search, Firefly has a lower average AUC of 55.12% and high specificity (99.12%). Firefly's low recall (11.11%) and F1 Score (16%) indicate that it neglects good examples despite its high precision. Harris Hawks has a modest AUC of 60.73% and an acceptable accuracy of 87.12%. With an F1 Score of 32.18%, it balances precision and recall. Its 97.02% specificity means it avoids false positives while retaining quality recall. The models with the lowest average AUC (52.08%), accuracy (20%), recall (5.56%), and F1 Score (8.64%) were Particle Swarm. Particle Swarm struggles with class difference and positive instance identification despite its 98.60% specificity. Cuckoo Search and Firefly have excellent specificity but lower recall and F1 scores than the OM model, which has better AUC, accuracy, and balance. Particle Swarm is less accurate at recognizing positives and balancing precision and recall than Harris Hawks. Figure 15 presents a diagram that compares the AUC values.

Cuckoo Search offers the best AUC, accuracy, precision, and recall, although each method has pros and cons. OM and Particle Swarm are more accurate but struggle with precision and memory, while Firefly and Harris Hawks are inconsistent and fail to balance detecting performance. The AUC values will be compared, as shown in Fig. 16.

See Table 12 for metaheuristic algorithm performance metrics on the MW1 dataset. OM exhibits a high accuracy of 88% and an average AUC of 60.03%. Although accurate, its precision (48.62%) and recall (14%) are poor, resulting in a mediocre F1 Score of 19.81%. The OM model is correct but struggles to balance positive case detection, which hinders its

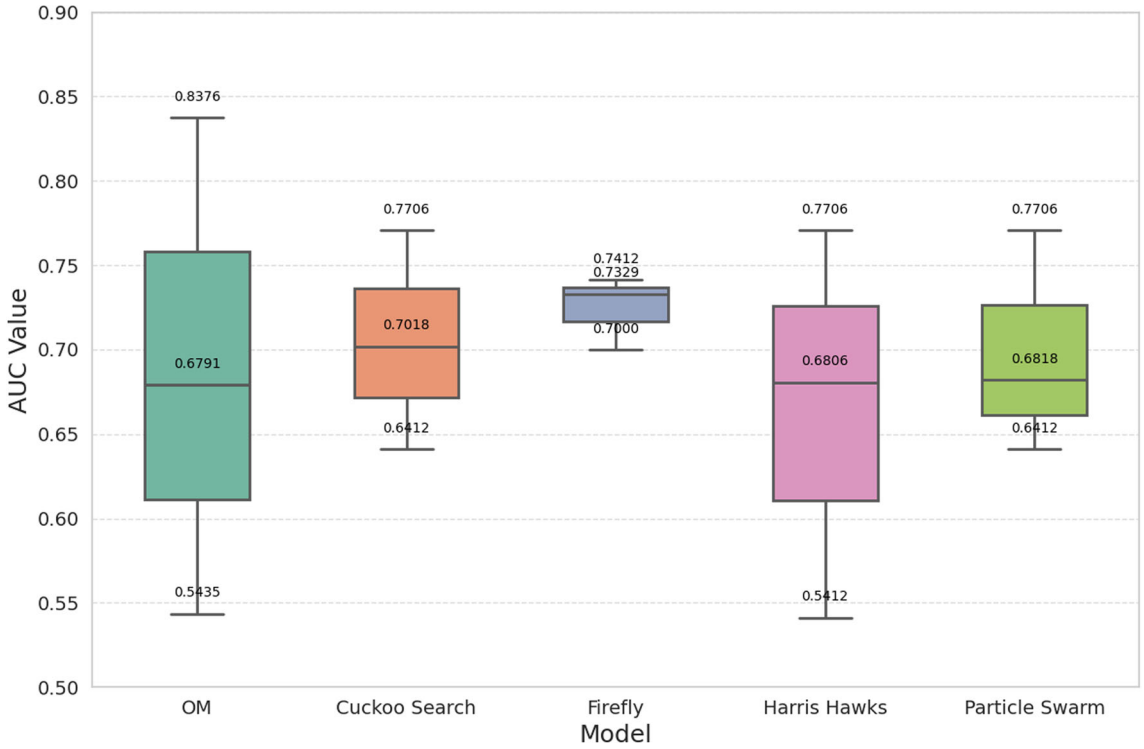

Fig. 13 AUC comparison between WSO and other metaheuristics on Ar4 dataset

Table 10 Comparison of WSO with other optimizers on Ar5 dataset

Dataset	Model	AUC				Accuracy Avg	Precision Avg	Recall Avg	F1 Score Avg	Specificity Avg
		Worst	Best	Avg	Std					
Ar5	OM	0.5600	0.9349	0.7467	8e-4	0.8200	0.8000	0.4533	0.5520	0.9280
	Cuckoo search	0.6667	0.8333	0.7900	7.21e−2	0.8375	0.9667	0.6000	0.7267	0.9800
	Firefly	0.5667	0.8333	0.7767	8.76e−2	0.8125	0.8500	0.6333	0.7200	0.9200
	Harris Hawks	0.5667	0.8333	0.6867	9.19e−2	0.7500	0.8667	0.4333	0.5567	0.9400
	Particle Swarm	0.5667	0.8333	0.6333	8.61e−2	0.7000	0.7500	0.3667	0.4800	0.9000

practical uses. Cuckoo Search ranks higher in AUC (74.59%) and accuracy (90.85%). Its F1 Score of 52.48% is more significant due to its more robust precision (53.94%) and recall (55%). Cuckoo Search may better distinguish positive and negative instances while balancing precision and recall. Cuckoo Search outperforms Firefly, which averages 56.40% AUC. Extreme precision (77.50%) and accuracy (81.57%) are offset by weak recall (15%) and a lower F1 Score of 21.61%. Firefly is accurate but misses many positive cases, resulting in poor performance. Harris Hawks has a lower AUC (57.13%) and accuracy (82.75%). It has an F1 Score of 24.22% due to higher precision (81%) and recall (15%) than Firefly. It has higher precision and recall than Firefly but less detection capacity. Particle Swarm had the highest accuracy at 82.75% with an average AUC of 56.38%. Its F1 Score is 22.47% due to its high precision (96.67%) and low recall (13%). Although Particle Swarm is exact, it fails to recognize affirmative cases, decreasing its performance.

WSO enhances classification performance and ensures model stability and consistency, evidenced by the reduced standard deviation across datasets. This is crucial, particularly in practical scenarios that necessitate dependability. This model exhibits stability due to optimization, in contrast to others. Results are derived from 15 benchmark datasets of varying complexities. The OM model demonstrated scalability, generalizability, and domain resilience. This demonstrates the model's versatility across many data types, rendering it effective for defect classification. The number of weak learners has not increased, but has decreased. Also, the search space has not expanded, but has shrunk. Consequently, the optimization process has become faster and more efficient. In conclusion, the findings and experiments demonstrate significant enhancements in classifica-

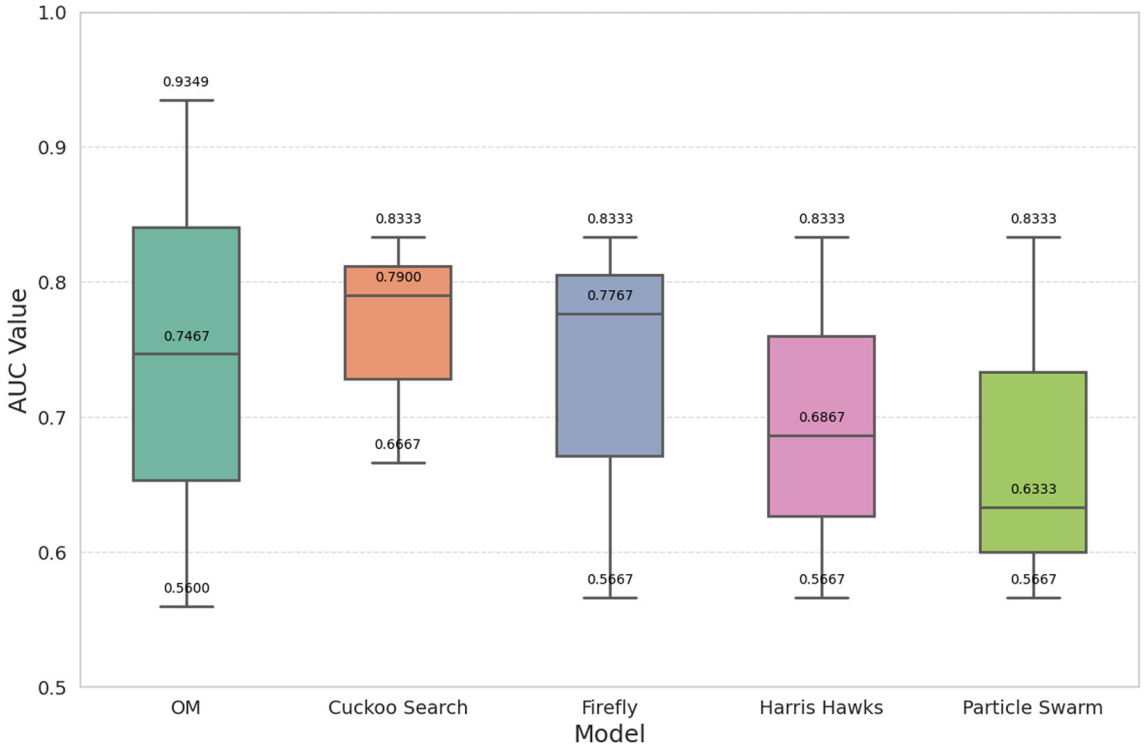

Fig. 14 AUC comparison between WSO and other metaheuristics on Ar5 dataset

Table 11 Comparison of WSO with other optimizers on CM1 dataset

Dataset	Model	AUC				Accuracy	Precision	Recall	F1 Score	Specificity
		Worst	Best	Avg	Std	Avg	Avg	Avg	Avg	Avg
CM1	OM	0.6243	0.8621	0.6984	1.0e−3	0.7950	0.3758	0.1513	0.1947	0.8937
	Cuckoo search	0.4912	0.6871	0.5898	7.04e−2	0.8652	0.4762	0.2111	0.2710	0.9684
	Firefly	0.5000	0.6579	0.5512	6.20e−2	0.8712	0.4179	0.1111	0.1600	0.9912
	Harris Hawks	0.5468	0.6871	0.6073	5.86e−2	0.8712	0.6017	0.2444	0.3218	0.9702
	Particle Swarm	0.4912	0.6023	0.5208	3.68e−2	0.8591	0.2000	0.0556	0.0864	0.9860

tion accuracy, scalability, generalization, and computational efficiency, comprehensively addressing software defect classification. The methodology and findings are substantial, refuting the assertion of insignificance. Based on the comparisons presented above, it is clear that OM is much more effective than the other algorithms that were utilized in the comparison in terms of the area under the curve (AUC). The fact that this is the case demonstrates that OM can benefit activities that demand excellent predictive performance and consistency, making it a feasible option for classification tasks.

For further analysis, Table 13 is conducted, which includes a comparative p-test between WSO and other metaheuristic algorithms.

Performance differences between WSO and other algorithms (CS, Firefly, HHO, and PSO) are not statistically significant, as shown in Table 13. All p values surpass 0.05, indicating that the methods perform similarly for Apache, Ar4, CM1, and MW1 in the examined situation. For dataset Ar5, the performance differences between WSO and the other algorithms (CS, Firefly, HHO, and PSO) are not statistically significant. Although not significant, WSO vs. HHO and WSO vs. PSO p-values approach the threshold, encouraging further inquiry.

5.7 Execution Time

Figure 17 shows the results of the execution time comparison between an optimized ensemble learning model (OM) and a standard ensemble learning model (EM) using the WSO metaheuristic algorithm. All employed datasets' execution times in this study are recorded, and each dataset is assessed using both EM and OM. The following provides a thorough analysis and discussion of the execution time obtained.

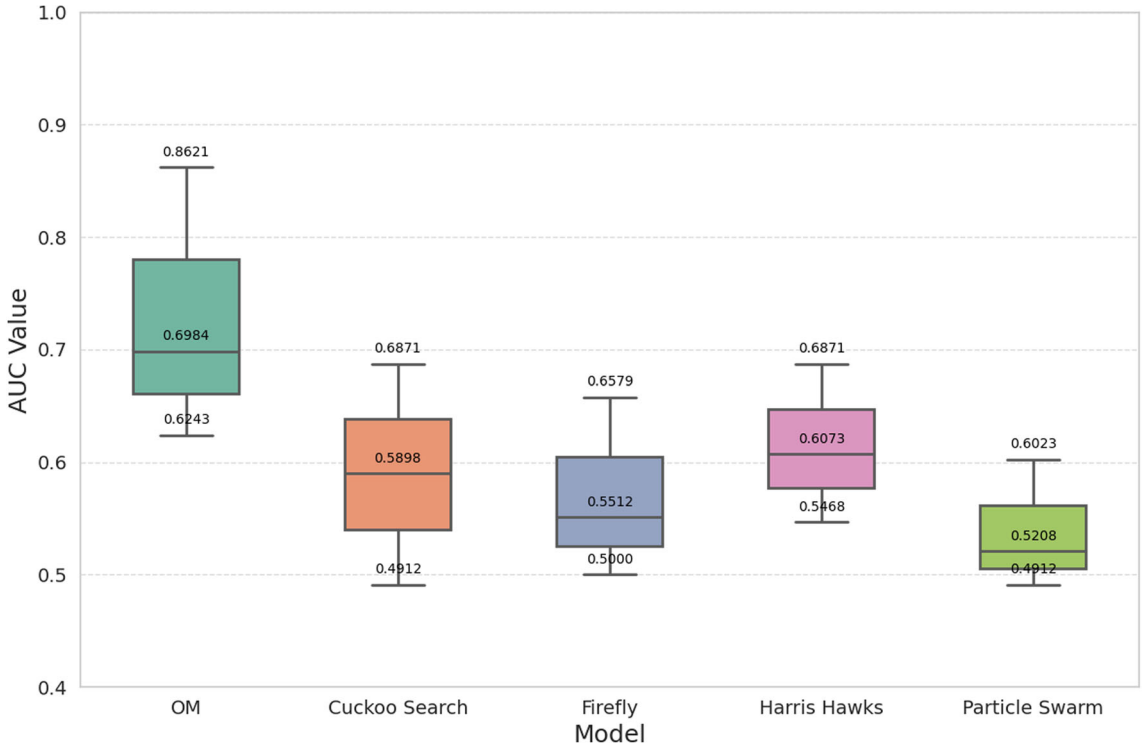

Fig. 15 AUC comparison between WSO and other metaheuristics on CM1 dataset

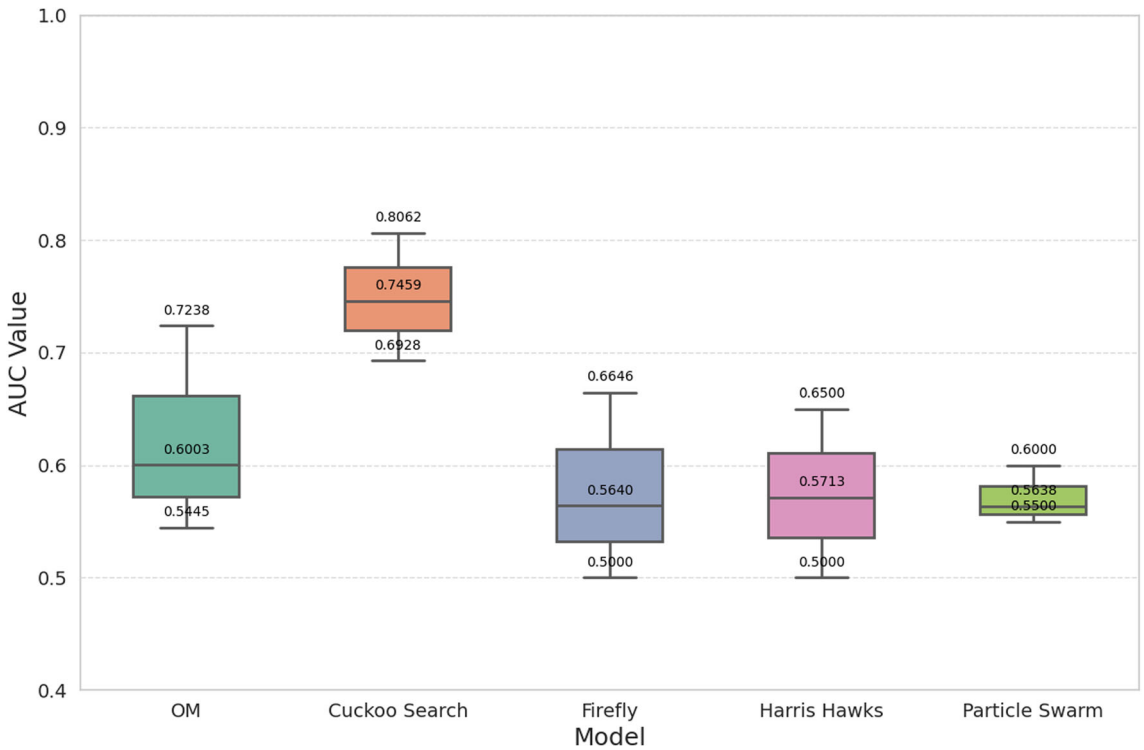

Fig. 16 AUC comparison between WSO and other metaheuristics on MW1 dataset

Table 12 Comparison of WSO with other optimizers on MW1 dataset

Dataset	Model	AUC				Accuracy	Precision	Recall	F1 Score	Specificity
		Worst	Best	Avg	Std	Avg	Avg	Avg	Avg	Avg
MW1	OM	0.5445	0.7238	0.6003	2e-4	0.8800	0.4862	0.1400	0.1981	0.9516
	Cuckoo search	0.6928	0.8062	0.7459	3.50e−2	0.9085	0.5394	0.5500	0.5248	0.9419
	Firefly	0.5000	0.6646	0.5640	4.29e−2	0.8157	0.7750	0.1500	0.2161	0.9780
	Harris Hawks	0.5000	0.6500	0.5713	4.57e−2	0.8275	0.8100	0.1500	0.2422	0.9927
	Particle Swarm	0.5500	0.6000	0.5638	2.24e−2	0.8275	0.9667	0.1300	0.2247	0.9976

Table 13 p-value comparative test for WSO and other metaheuristic algorithms

Dataset	WSO vs CS	WSO vs Firefly	WSO vs HHO	WSO vs PSO
Apache	6.25E−02	6.25E−02	1.25E−01	6.25E−02
Ar4	6.25E−02	6.25E−02	6.25E−02	6.25E−02
Ar5	4.38E−01	1.25E−01	6.25E−02	6.25E−02
CM1	6.25E−02	6.25E−02	6.25E−02	6.25E−02
MW1	6.25E−02	6.25E−02	6.25E−02	6.25E−02

Figure shows two entries for each dataset: one for EM and one for OM. The times of execution are expressed in seconds. Shorter execution durations indicate optimized processing efficiency. The execution time of OM is consistently faster than that of EM across all datasets. This suggests that the optimized OM model with the WSO metaheuristic algorithm for all investigated datasets delivers speedier execution times than the regular EM model.

For both models (EM and OM), there is a notable variation in execution times between datasets. The efficacy of the optimization algorithm may vary depending on the dataset's features, as some datasets exhibit more significant execution time variations between EM and OM.

The efficiency of the WSO algorithm in optimizing the ensemble learning model by defining the adequate number of weak learners is demonstrated by the notable decrease in execution time for OM when compared to EM. Optimization techniques like WSO facilitate the discovery of more effective solutions by allowing one to explore the search area and make more intelligent decisions on the ensemble model's structure and weak learner numbers.

Owing to its quicker execution speeds, OM is more suited for time-sensitive or real-time applications where making decisions quickly is essential. The optimized OM model becomes more scalable and cost-effective when execution durations are reduced, especially in contexts with resource-constrained environments or large-scale issues.

Finally, the execution time results show how well the WSO optimizer performs when optimizing ensemble learning models, resulting in noticeably faster execution times for various situations. This demonstrates how optimization strategies can enhance model scalability and computing efficiency. Table 14 compares execution time before and after optimization.

5.8 BoxPots of EM and OM Models

Following a binary-WSO algorithm's optimization of the ensemble learning model, boxplots, a technique for visualizing the results, are commonly used to present and compare the performance outcomes. A boxplot, illustrated in Fig. 18, shows essential statistics like the median, quartiles, and possible outliers clearly and succinctly, summarizing the distribution of results from multiple independent experimental runs.

The boxplot helps show how the optimization process affects the model's performance across many measures while optimizing an ensemble learning model. Several conclusions can be drawn from contrasting the outcomes of the optimized model with those of the standard model (EM).

First, compared to the EM model, the optimized version of the ML model (OM) frequently shows better performance characteristics. Metrics like accuracy, precision, recall, F1-score, and area under the curve (AUC) for classification tasks can all be used to track this improvement. Second, when comparing the optimized model to the conventional model, the distribution of performance measures across several runs and datasets could seem more condensed and uniform. More minor variations in the whiskers' lengths and narrower interquartile ranges (IQRs) indicate this in the boxplots.

Furthermore, the OM can show fewer outliers or extreme values in the boxplot than the EM. This suggests optimization has made performance across various scenarios or datasets more stable and dependable. In addition, the developed OM

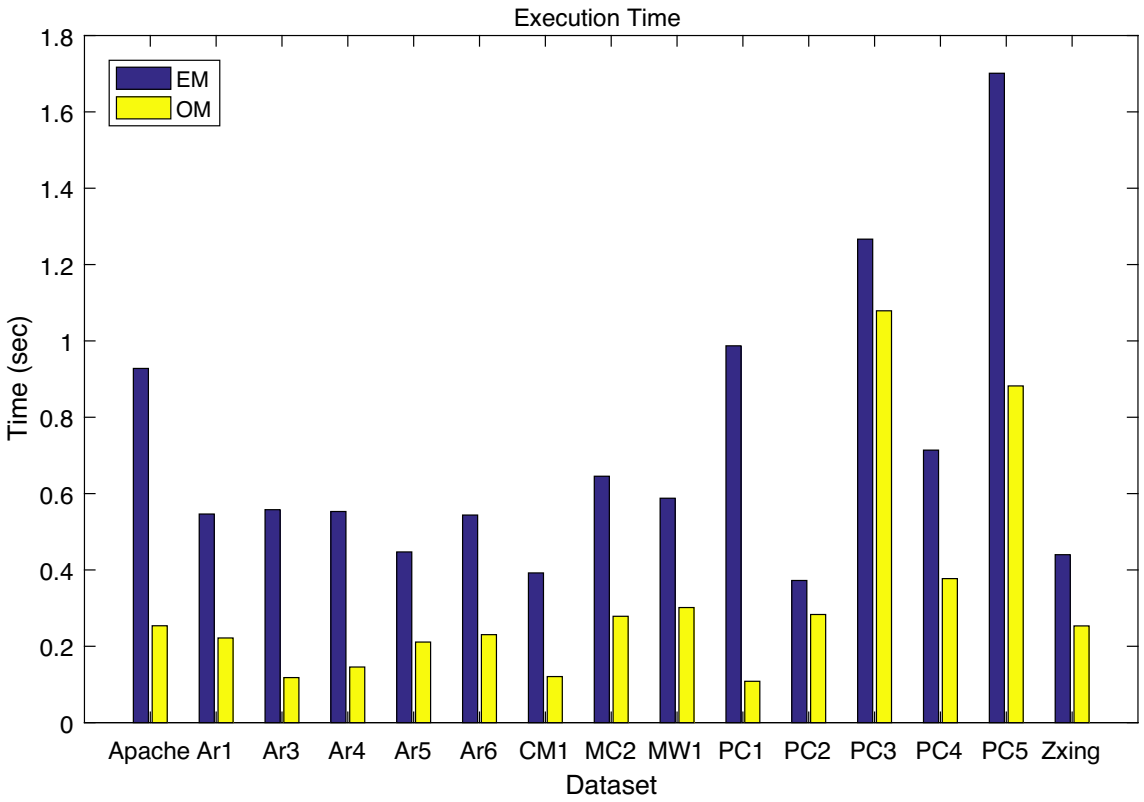

Fig. 17 Execution time of standard ensemble learning and optimized ensemble learning model for different datasets

Table 14 Execution time before and after optimization in term of seconds

Project	Before optimization	After optimization
Zxing	6.98E+00	5.82E−01
Apache	4.40E+00	2.90E+00
PC1	1.03E+01	2.63E+00
PC2	8.01E+00	1.60E+00
PC3	1.68E+01	1.08E+00
PC4	1.00E+01	8.07E−01
PC5	2.63E+01	8.68E+00
Ar1	2.18E+00	3.59E−01
Ar3	2.15E+00	2.63E−01
Ar4	2.17E+00	6.75E−01
Ar5	2.18E+00	3.53E−01
Ar6	4.46E+00	3.06E−01
CM1	8.22E+00	9.98E−01
MC2	2.63E+00	9.55E−01
MW1	2.26E+00	5.05E−01

typically has more shrinking boxplots, that is, boxplots with fewer widths than the EM. This shows that the optimization process has successfully adjusted the EM model's weak learners number to improve overall performance while lowering variability across different runs and all used datasets.

As shown in Fig. 18, the boxplot visualizes the OM model's performance outcomes and offers insightful information about how well the binary-WSO algorithm's optimization procedure works. It facilitates understanding the improvements made to the created model's performance and how it stacks up against the standard model's baseline performance.

6 Conclusion and Future Work

This paper has presented an intelligent, optimized ensemble-based approach combining a range of classifiers, making significant advances within the domain of software fault prediction. We aimed to improve the accuracy, dependability, and effectiveness of defect prediction models by utilizing ML techniques and the White Shark Optimizer (WSO) metaheuristic algorithm.

The methodology developed in this study proposes a novel strategy that integrates ensemble learning with the WSO optimizer too to enhance the accuracy of defect prediction models. Through the optimization of ensemble classifier numbers, superior prediction performance is attained com-

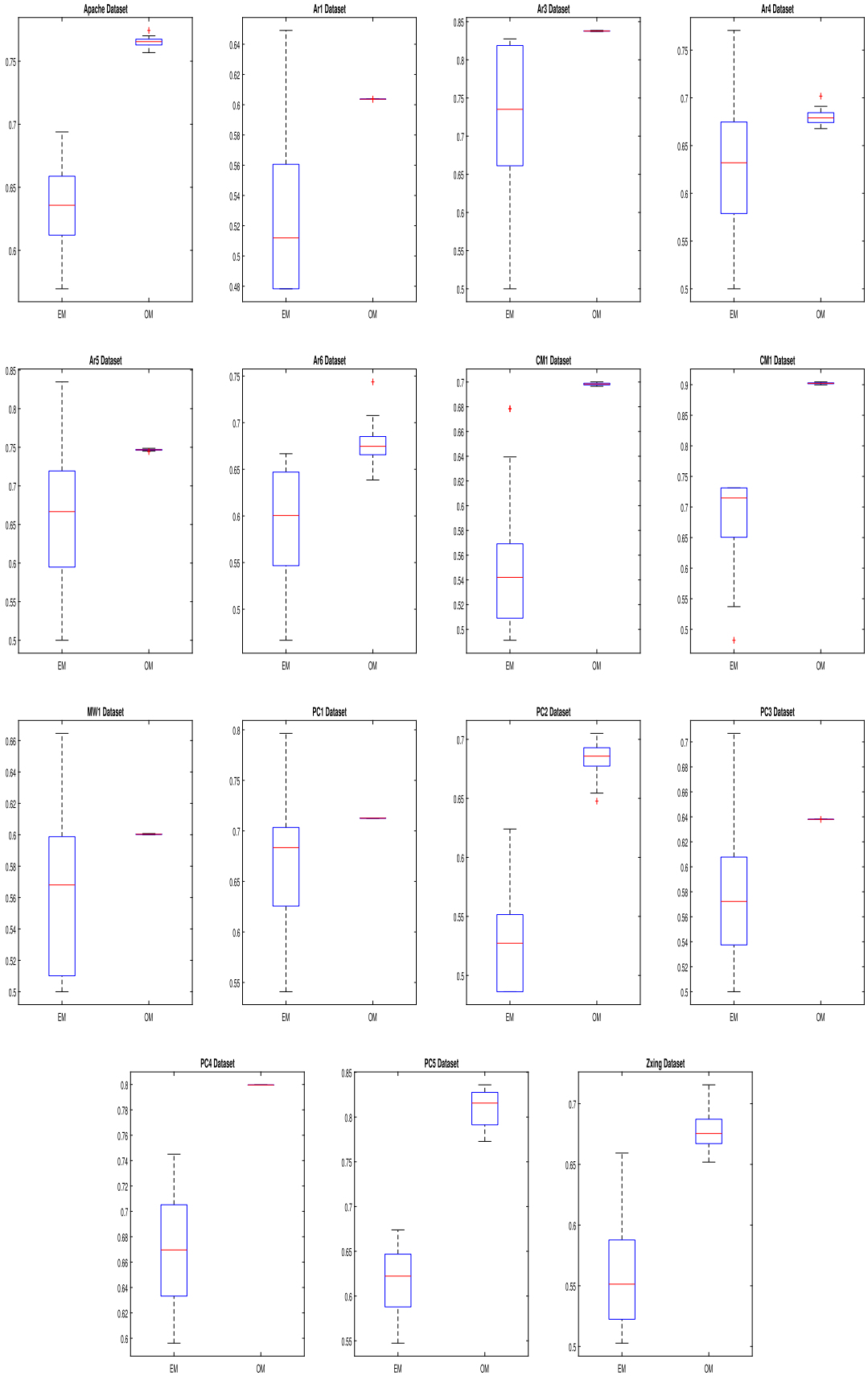

Fig. 18 Boxplot comparison of EM and OM models over used dataset

pared to conventional ensemble methods. The results indicate that the optimized ensemble scheme consistently performs better than the standard ensemble model across various performance metrics, including AUC-ROC, Accuracy, Precision, Recall, F1-score, and Specificity. The observed increase in performance serves as evidence for the efficacy of our proposed methodology in augmenting the dependability and accuracy of defect prediction models. The consistent performance benefits observed across several benchmark datasets serve as evidence for the scalability and generalizability of the developed strategy. The optimized ensemble model developed, utilizing the WSO algorithm, demonstrates its efficacy in addressing diverse software defect prediction issues. It exhibits consistent performance and adaptability across different benchmark datasets.

The results of the investigations demonstrate that adopting ensemble methods, specifically when optimized by applying metaheuristic algorithms such as WSO, can yield substantial enhancements in the predictive capabilities of defect prediction models. The developed approach enhances accuracy and reliability in identifying faulty software modules by integrating various classifiers. Utilizing the WSO optimizer yields consistent improvements in performance across multiple datasets and performance measures. By enhancing the overall prediction capabilities of the ensemble model, the optimization procedure minimizes variability in model performance.

The limitations of this study are summarized as follows: The suggested approach demonstrates efficacy across all benchmark datasets; however, the degree of performance enhancement may vary according to the dataset's specific characteristics. Certain datasets may benefit more from optimization methods than others, indicating a degree of interdependence among them. Applying WSO in optimization requires multiple iterations to ascertain the ideal quantity of weak classifiers, which may increase computational complexity and time requirements. The optimization of computer resources for predictive performance is a vital consideration in practical applications. In addition, enhancing the optimal white sharks solely through their mathematical model is inadequate. Furthermore, the adaptive parameter P_1 progressively diminishes with time, reducing the capacity of the optimal white shark to direct all white sharks and enhancing the exploitation potential of WSO during the last phases of the searching process. They also encounter challenges in assisting white sharks in evading adjacent optimal. Enhancing WSO's exploitation capabilities during the last phases of the search process is essential. This mitigates suffering from premature convergence while enhancing the convergence rate of the WSO method.

For future purposes, subsequent investigations may use the proposed methodology on supplementary datasets, software prediction challenges, and broader prediction and classification engineering issues. Examining the influence of various ensemble configurations and optimization strategies on predictive performance can yield further knowledge regarding the efficacy of defect prediction methodologies. The use of defect prediction models in software development processes has the potential to improve proactive defect management and quality assurance for software projects. Subsequent research endeavors may prioritize advancing approaches to achieve smooth integration and continuous monitoring of defect prediction models inside software development processes. We could further enhance predictive performance by integrating ensemble learning with other optimization approaches or hybridizing other defect prediction methodologies. Hybrid methodologies that capitalize on the respective advantages of various techniques have the potential to provide improved levels of accuracy and reliability in the prediction of defects. In addition, the demonstrated efficacy of the proposed binary variants of WSO in software fault classification allows for their reimplementation in many issues characterized by a binary search space. Furthermore, its performance may be enhanced by employing additional efficient components from alternative metaheuristic algorithms. Ultimately, additional real-world SDP datasets may be utilized for forthcoming research endeavors.

Threats to validity

The study's design and methods may affect internal validity. The dataset may be inconsistent, or developer experience or software complexity metrics that affect fault classification were not recorded. We used NASA, softlab, and Relin records from defect prediction research to adjust for variations. However, unobserved factors may affect defect classification. For external validity difficulties, our study's findings may not apply to all software systems or sectors, because the datasets used to train and test the models may not represent real-world software initiatives. Even though it worked well in our research, the White Shark Optimizer may not work well in other scenarios or datasets. The authors used widely recognized benchmark datasets for fault prediction. Testing on more diverse and current datasets is needed to prove the model's real-world applicability. Conclusion validity was assessed by comparing the model to metaheuristic models and utilizing statistical significance tests on distinct data subsets. Since WSO optimization speed enhancements are dataset-specific, resilience requires additional testing.

Appendix A: Nasa, Relink, and Softlab Datasets

Each group of the datasets used contains a unique collection of characteristics. Consequently, to address and declare these properties, Tables 15, 16, and 17 provide a concise breakdown of the nature of the associated attribute for each group of the dataset individually.

Appendix B

See Table 15.

Table 15 Nasa datasets features declaration

Metrics	Description
LOCBLANK	Number blank lines in the module
BRANCHCOUNT	Number branches(e.g., if statements, switch cases) in the module
CALLPAIRS	Number call pairs (e.g., function calls) in the module
LOCCODEANDCOMMENT	Number lines containing both code and comments in the module
LOCCOMMENTS	Number lines containing comments in the module
CONDITIONCOUNT	Number conditions (e.g., Boolean expressions) in the module
CYCLOMATICCOMPLEXITY	Number linearly independent paths through the code
CYCLOMATICDENSITY	The ratio of cyclomatic complexity to lines of code (LOC)
DECISIONCOUNT	Number decision points (e.g., if conditions, switch cases) in the module
DESIGNCOMPLEXITY	The overall complexity of the module's design
DESIGNDENSITY	The ratio of design complexity to lines of code (LOC)
EDGECOUNT	Number of edges in the module's control flow graph
ESSENTIALCOMPLEXITY	Complexity inherent in the problem that must be addressed by the software
ESSENTIALDENSITY	Representing the ratio of essential complexity to lines of code (LOC)
LOCEXECUTABLE	Number lines containing executable code in the module
PARAMETERCOUNT	Number formal parameters (e.g., function arguments) in the module
GLOBALDATACOMPLEXITY	Complexity related to global data structures in the module
GLOBALDATADENSITY	Density of global data structures in the module
HALSTEADCONTENT	Representing the information content of the program
HALSTEADDIFFICULTY	Representing the ease of understanding the program
HALSTEADEFFORT	Indicating the projected program comprehension and implementation time
HALSTEADERROREST	Representing the number of predicted errors in the program
HALSTEADLENGTH	Total number of (operators+operands)in the module
HALSTEADLEVEL	The level of difficulty in understanding the program
HALSTEADPROGTIME	The estimated time to implement the program
HALSTEADVOLUME	The program size or the total number of bits required to encode the program
MAINTENANCESEVERITY	Severity of maintenance required for the module
MODIFIEDCONDITIONCOUNT	Number modified conditions (e.g., ternary operators) in the module
MULTIPLECONDITIONCOUNT	Nested Boolean expressions in the module should be numbered
NODECOUNT	Number nodes in the module's control flow graph
NORMALIZEDCC	Representing a normalized version of cyclomatic complexity
NUMOPERANDS	Number of operands in the module
NUMOPERATORS	Total number of operators in the module
NUMUNIQUEOPERANDS	Number unique operands (variables, constants, etc.) in the module
NUMUNIQUEOPERATORS	Number unique operators (arithmetic, logical, etc.) in the module
NUMBEROFLINES	Total number of lines in the module
PERCENTCOMMENTS	Percentage of lines containing comments in the module
LOCTOTAL	Total lines of code (LOC) in the module
Defective	Binary label indicating whether the module contains defects (1) or not (0)

Appendix C

See Table 16.

Table 16 Relink datasets features declaration

Metrics	Description
AvgCyclomatic	Average cyclomatic complexity for all nested functions of methods
AvgCyclomaticStrict	Logical ANDs and ORs in conditional expressions
AvgEssential	The average complexity inherent in a problem that must be addressed
AvgLine	Average number of lines for all nested functions or methods
AvgLineBlank	Calculates the average number of blank lines within the module
AvgLineCode	The average number of lines containing executable code within the module
AvgLineComment	Calculates the average number of lines containing comments within the module
CountLine	Number of all lines for all nested functions or methods
CountLineBlank	Counts the total number of blank lines within the module
CountLineCode	Counts the total number of lines containing executable code within the module
CountLineCodeDecl	Counts the total number of lines with code declarations within the module
CountLineCodeExe	Counts the total number of lines with executable code within the module
CountLineComment	Counts the total number of lines with comments within the module
CountSemicolon	Counts the total number of semicolons within the module
CountStmt	Counts the total number of statements within the module
CountStmtDecl	Counts the total number of statements with declarations within the module
CountStmtExe	Counts the total number of executable statements within the module
MaxCyclomatic	Maximum cyclomatic complexity of all nested functions or methods
MaxCyclomaticModified	Represents a modified version of the maximum cyclomatic complexity
MaxCyclomaticStrict	The module's maximum nested function or method cyclomatic complexity
RatioCommentToCode	Maximum cyclomatic complexity of all nested functions or methods
SumCyclomatic	Sum of cyclomatic complexity of all nested functions or methods
SumCyclomaticModified	Represents a modified version of the sum of cyclomatic complexity
SumCyclomaticStrict	A module's restricted sum of nested functions or methods cycle complexity
SumEssential	Indicates the module's essential complexity for all nested functions or methods
Defective	Binary label indicating whether the module contains defects (1) or not (0)

Appendix D

See Table 17.

Table 17 Softlab datasets features declaration

Metrics	Description
Totalloc	Total lines of code (LOC) in the module
Blankloc	Number blank lines in the module
Commentloc	Number lines containing comments in the module
Codeandcommentloc	Number lines containing both code and comments in the module
Executableloc	Number lines containing executable code in the module
Uniqueoperands	Number unique operands (variables, constants, etc.) in the module
Uniqueoperators	Number unique operators (arithmetic, logical, etc.) in the module
Totaloperands	Total number of operands in the module
Totaloperators	Total number of operators in the module
Halsteadvocabulary	Representing the number of unique operators and operands in the module
Halsteadlength	Total operator and operand occurrences in module
Halsteadvolume	Reflecting the program size or total bits needed to encode it
Halesteadlevel	Indicating the level of difficulty in understanding the program
Halsteaddifficulty	Representing the ease of understanding the program
Halsteadeffort	Indicating the projected program comprehension and implementation time
Halsteaderror	Representing the number of predicted errors in the program
Halsteadtime	Representing the estimated time to implement the program
Branchcount	Number branches (e.g., if statements, switch cases) in the module
Decisioncount	Number decision points (e.g., if conditions, switch cases) in the module
Callpairs	Number call pairs (e.g., function calls) in the module
Conditioncount	Number conditions (e.g., Boolean expressions) in the module
Multipleconditioncount	Nested Boolean expressions in the module should be numbered
Cyclomaticcomplexity	Representing the number of linearly independent paths through the code
Cyclomaticdensity	Representing the ratio of cyclomatic complexity to LOC
Decisiondensity	Representing the ratio of decision points to LOC
Designcomplexity	Representing the overall complexity of the module's design
Designdensity	Representing the ratio of design complexity to LOC
Normalizedcyclomaticcomplexity	Representing a normalized version of cyclomatic complexity
Formalparameters	Number formal parameters (e.g., function arguments) in the module
Defective	Binary label indicating whether the module contains defects (1) or not (0)

Acknowledgements The authors received no financial support for this article's research, authorship, and publication. The researchers will incur the expenses associated with publishing this research paper.

Author Contributions All authors participated equally in the preparation of this research in terms of preparation, data preprocessing, and technical experiments.

Data Availability Data is available upon request.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Madhavaram, S., Appan, R., Manis, K., Browne, G.J.: Building capabilities for software development firm competitiveness: the role of intellectual capital and intra-firm relational capital. Inf. Manag. **60**(2), 103744 (2023). https://doi.org/10.1016/j.im.2022.103744
2. Holgeid, K.K., Jorgensen, M., Sjoberg, D.I., Krogstie, J.: Benefits management in software development: a systematic review of empirical studies. IET Softw. **15**(1), 1–24 (2021). https://doi.org/10.1049/sfw2.12007
3. Singh, V., Kumar, V., Singh, V.: A hybrid novel fuzzy ahp-topsis technique for selecting parameter-influencing testing in software development. Decis. Anal. J. **6**, 100159 (2023). https://doi.org/10.1016/j.dajour.2022.100159
4. Cai, C.-H., Sun, J., Dobbie, G.: B model quality assessments on automated reachability repair with iso/iec 25010. Sci. Comput. Program. **214**, 102732 (2022). https://doi.org/10.1016/j.scico.2021.102732
5. Borstler, J., Bennin, K.E., Hooshangi, S., Jeuring, J., Keuning, H., Kleiner, C., MacKellar, B., Duran, R., Storrle, H., Toll, D., et al.: Developers talking about code quality. Empir. Softw. Eng. **28**(6), 128 (2023). https://doi.org/10.1007/s10664-023-10381-0
6. Al Dallal, J., Abdulsalam, H., AlMarzouq, M., Selamat, A.: Machine learning-based exploration of the impact of move method refactoring on object-oriented software quality attributes. Arab. J. Sci. Eng. (2023). https://doi.org/10.1007/s13369-023-08174-0
7. Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., Babar, M.A.: Understanding and addressing quality attributes of microservices architecture: a systematic literature review. Inf. Softw. Technol. **131**, 106449 (2021). https://doi.org/10.1016/j.infsof.2020.106449
8. Yaghoobi, T.: Selection of optimal software reliability growth model using a diversity index. Soft. Comput. **25**(7), 5339–5353 (2021). https://doi.org/10.1007/s00500-020-05532-0
9. Saxena, P., Kumar, V., Ram, M.: A novel critic-topsis approach for optimal selection of software reliability growth model (srgm). Qual. Reliab. Eng. Int. **38**(5), 2501–2520 (2022). https://doi.org/10.1002/qre.3087
10. Huang, Y.-S., Chiu, K.-C., Chen, W.-M.: A software reliability growth model for imperfect debugging. J. Syst. Softw. **188**, 111267 (2022). https://doi.org/10.1016/j.jss.2022.111267
11. Verma, V., Anand, S., Aggarwal, A.G.: Optimal time for management review during testing process: an approach using s-curve two-dimensional software reliability growth model. Int. J. Qual. Reliab. Manag. (2023). https://doi.org/10.1108/IJQRM-08-2022-0236
12. Panwar, S., Kumar, V., Kapur, P., Singh, O.: Software reliability prediction and release time management with coverage. Int. J. Qual. Reliab. Manag. **39**(3), 741–761 (2022). https://doi.org/10.1108/IJQRM-05-2021-0139
13. Pradhan, V., Dhar, J., Kumar, A.: Software reliability models and multi-attribute utility function based strategic decision for release time optimization. Predict. Anal. Syst. Reliab. (2022). https://doi.org/10.1007/978-3-031-05347-4_12
14. Chen, X., Xia, H., Pei, W., Ni, C., Liu, K.: Boosting multi-objectie just-in-time software defect prediction by fusing expert metrics and semantic metrics. J. Syst. Softw. **206**, 111853 (2023). https://doi.org/10.1016/j.jss.2023.111853
15. Hameed, S., Elsheikh, Y., Azzeh, M.: An optimized case-based software project effort estimation using genetic algorithm. Inf. Softw. Technol. **153**, 107088 (2023). https://doi.org/10.1016/j.infsof.2022.107088
16. Pai, A.R., Joshi, G., Rane, S.: Quality and reliability studies in software defect management: a literature review. Int. J. Qual. Reliab. Manag. **38**(10), 2007–2033 (2021). https://doi.org/10.1108/IJQRM-07-2019-0235
17. Sreekanth, N., Rama Devi, J., Shukla, K.A., Mohanty, D., Srinivas, A., Rao, G.N., Alam, A., Gupta, A.: Evaluation of estimation in software development using deep learning-modified neural network. Appl. Nanosci. **13**(3), 2405–2417 (2023). https://doi.org/10.1007/s13204-021-02204-9
18. Ritu, Bhambri, P.: Software effort estimation with machine learning–a systematic literature review. In: Agile software development: trends, challenges and applications, pp. 291–308. Wiley Online Library (2023) https://doi.org/10.1002/9781119896838.ch15
19. Manoj, N., Deepak, G.: Sdpo: An approach towards software defect prediction using ontology driven intelligence. In: International Conference on Electrical and Electronics Engineering, pp. 164–172. Springer (2022). https://doi.org/10.1007/978-981-19-1677-9_15
20. Qiao, L., Li, X., Umer, Q., Guo, P.: Deep learning based software defect prediction. Neurocomputing **385**, 100–110 (2020). https://doi.org/10.1016/j.neucom.2019.11.067
21. Kanwar, S., Awasthi, L.K., Shrivastava, V.: Candidate project selection in cross project defect prediction using hybrid method. Expert Syst. Appl. **218**, 119625 (2023). https://doi.org/10.1016/j.eswa.2023.119625
22. Giray, G., Bennin, K.E., Koksal, O., Babur, O., Tekinerdogan, B.: On the use of deep learning in software defect prediction. J. Syst. Softw. **195**, 111537 (2023). https://doi.org/10.1016/j.jss.2022.111537
23. Khatri, Y., Singh, S.K.: An effective feature selection based cross-project defect prediction model for software quality improvement. Int. J. Syst. Assur. Eng. Manag. **14**(Suppl 1), 154–172 (2023). https://doi.org/10.1007/s13198-022-01831-x

24. Jing, X.-Y., Chen, H., Xu, B.: Cross-project defect prediction. In: Intelligent Software Defect Prediction, pp. 35–63. Springer (2024)
25. Zheng, W., Shen, T., Chen, X., Deng, P.: Interpretability application of the just-in-time software defect prediction model. J. Syst. Softw. **188**, 111245 (2022). https://doi.org/10.1016/j.jss.2022.111245
26. Goyal, S.: Effective software defect prediction using support vector machines (svms). Int. J. Syst. Assur. Eng. Manag. **13**(2), 681–696 (2022). https://doi.org/10.1007/s13198-021-01326-1
27. Awotunde, J.B., Misra, S., Adeniyi, A.E., Abiodun, M.K., Kaushik, M., Lawrence, M.O.: A feature selection-based k-nn model for fast software defect prediction. In: International Conference on Computational Science and Its Applications, pp. 49–61. Springer (2022). https://doi.org/10.1007/978-3-031-10542-5_4
28. Cetiner, M., Sahingoz, O.K.: A comparative analysis for machine learning based software defect prediction systems. In: 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–7. IEEE (2020).https://doi.org/10.1109/ICCCNT49239.2020.9225352
29. Alaswad, F., Poovammal, E.: Software quality prediction using machine learning. Mater. Today: Proc. **62**, 4714–4720 (2022). https://doi.org/10.1016/j.matpr.2022.03.165
30. Khleel, N.A.A., Nehez, K.: Software defect prediction using a bidirectional lstm network combined with oversampling techniques. Clust. Comput. (2023). https://doi.org/10.1007/s10586-023-04170-z
31. Liu, J., Lei, J., Liao, Z., He, J.: Software defect prediction model based on improved twin support vector machines. Soft Comput. (2023). https://doi.org/10.1007/s00500-023-07984-6
32. Chennappan, R., et al.: An automated software failure prediction technique using hybrid machine learning algorithms. J. Eng. Res. **11**(1), 100002 (2023). https://doi.org/10.1016/j.jer.2023.100002
33. Di Sorbo, A., Zampetti, F., Visaggio, A., Di Penta, M., Panichella, S.: Automated identification and qualitative characterization of safety concerns reported in uav software platforms. ACM Trans. Softw. Eng. Methodol. **32**(3), 1–37 (2023). https://doi.org/10.1145/3564821
34. Li, N., Shepperd, M., Guo, Y.: A systematic review of unsupervised learning techniques for software defect prediction. Inf. Softw. Technol. **122**, 106287 (2020). https://doi.org/10.1016/j.infsof.2020.106287
35. Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., Aljaaf, A.J.: A systematic review on supervised and unsupervised machine learning algorithms for data science. In: Supervised and unsupervised learning for data science, pp. 3–21. Springer (2020). https://doi.org/10.1007/978-3-030-22475-2_1
36. Alsaeedi, A., Khan, M.Z.: Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. J. Softw. Eng. Appl. **12**(5), 85–100 (2019). https://doi.org/10.4236/jsea.2019.125007
37. Pemmada, S.K., Behera, H., Nayak, J., Naik, B.: Correlation-based modified long short-term memory network approach for software defect prediction. Evol. Syst. **13**(6), 869–887 (2022). https://doi.org/10.1007/s12530-022-09423-7
38. Bayoudh, K.: A survey of multimodal hybrid deep learning for computer vision: architectures, applications, trends, and challenges. Inf. Fusion (2023). https://doi.org/10.1016/j.inffus.2023.102217
39. Mahdieh, M., Mirian-Hosseinabadi, S.-H., Mahdieh, M.: Test case prioritization using test case diversification and fault-proneness estimations. Autom. Softw. Eng. **29**(2), 50 (2022). https://doi.org/10.1007/s10515-022-00344-y
40. Shah, M., Kantawala, H., Gandhi, K., Patel, R., Patel, K.A., Kothari, A.: Theoretical evaluation of ensemble machine learning techniques. In: 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 829–837. IEEE (2023). https://doi.org/10.1109/ICSSIT55814.2023.10061139
41. Mohammed, A., Kora, R.: A comprehensive review on ensemble deep learning: opportunities and challenges. J. King Saud Univ. Comput. Inf. Sci. (2023). https://doi.org/10.1016/j.jksuci.2023.01.014
42. Talaei Khoei, T., Ould Slimane, H., Kaabouch, N.: Deep learning: systematic review, models, challenges, and research directions. Neural Comput. Appl. **35**(31), 23103–23124 (2023). https://doi.org/10.1007/s00521-023-08957-4
43. Abou El Houda, Z., Brik, B., Ksentini, A.: Securing iiot applications in 6g and beyond using adaptive ensemble learning and zero-touch multi-resource provisioning. Comput. Commun. **216**, 260–273 (2024). https://doi.org/10.1016/j.comcom.2024.01.018
44. Batista, T., Bedregal, B., Moraes, R.: Constructing multi-layer classifier ensembles using the choquet integral based on overlap and quasi-overlap functions. Neurocomputing **500**, 413–421 (2022). https://doi.org/10.1016/j.neucom.2022.05.080
45. Kumari, S., Kumar, D., Mittal, M.: An ensemble approach for classification and prediction of diabetes mellitus using soft voting classifier. Int. J. Cogn. Comput. Eng. **2**, 40–46 (2021). https://doi.org/10.1016/j.ijcce.2021.01.001
46. Uddin, M.G., Nash, S., Rahman, A., Olbert, A.I.: Performance analysis of the water quality index model for predicting water state using machine learning techniques. Process Saf. Environ. Prot. **169**, 808–828 (2023). https://doi.org/10.1016/j.psep.2022.11.073
47. Ganie, S.M., Dutta Pramanik, P.K., Mallik, S., Zhao, Z.: Chronic kidney disease prediction using boosting techniques based on clinical parameters. PLoS ONE **18**(12), 0295234 (2023). https://doi.org/10.1371/journal.pone.0295234
48. Mafarja, M., Thaher, T., Al-Betar, M.A., Too, J., Awadallah, M.A., Abu Doush, I., Turabieh, H.: Classification framework for faulty-software using enhanced exploratory whale optimizer-based feature selection scheme and random forest ensemble learning. Appl. Intell. **53**(15), 18715–18757 (2023). https://doi.org/10.1007/s10489-022-04427-x
49. Al-Laham, M., Kassaymeh, S., Al-Betar, M.A., Makhadmeh, S.N., Albashish, D., Alweshah, M.: An efficient convergence-boosted salp swarm optimizer-based artificial neural network for the development of software fault prediction models. Comput. Electr. Eng. **111**, 108923 (2023). https://doi.org/10.1016/j.compeleceng.2023.108923
50. Tang, Y., Dai, Q., Yang, M., Du, T., Chen, L.: Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm. Int. J. Mach. Learn. Cybern. **14**(6), 1967–1987 (2023). https://doi.org/10.1007/s13042-022-01740-2
51. Raamesh, L., Jothi, S., Radhika, S.: Enhancing software reliability and fault detection using hybrid brainstorm optimization-based lstm model. IETE J. Res. **69**(12), 8789–8803 (2023). https://doi.org/10.1080/03772063.2022.2069603
52. Mondal, S., Sahu, A.K., Kumar, H., Pattanayak, R.M., Gourisaria, M.K., Das, H.: Software fault prediction using wrapper based ant colony optimization algorithm for feature selection. In: 2023 6th International Conference on Information Systems and Computer Networks (ISCON), pp. 1–6. IEEE (2023). https://doi.org/10.1109/ISCON57294.2023.10111995
53. Pandit, M.B.R., Varma, N.: A deep introduction to ai based software defect prediction (sdp) and its current challenges. In: TENCON 2019-2019 IEEE Region 10 Conference (TENCON), pp. 284–290. IEEE (2019). https://doi.org/10.1109/TENCON.2019.8929661
54. Mustaqeem, M., Saqib, M.: Principal component based support vector machine (pc-svm): a hybrid technique for software defect

detection. Clust. Comput. **24**(3), 2581–2595 (2021). https://doi.org/10.1007/s10586-021-03282-8
55. Jin, C.: Cross-project software defect prediction based on domain adaptation learning and optimization. Expert Syst. Appl. **171**, 114637 (2021). https://doi.org/10.1016/j.eswa.2021.114637
56. Alghanim, F., Azzeh, M., El-Hassan, A., Qattous, H.: Software defect density prediction using deep learning. IEEE Access **10**, 114629–114641 (2022). https://doi.org/10.1109/ACCESS.2022.3217480
57. Goyal, S.: Handling class-imbalance with knn (neighbourhood) under-sampling for software defect prediction. Artif. Intell. Rev. **55**(3), 2023–2064 (2022). https://doi.org/10.1007/s10462-021-10044-w
58. Alsawalqah, H., Hijazi, N., Eshtay, M., Faris, H., Radaideh, A.A., Aljarah, I., Alshamaileh, Y.: Software defect prediction using heterogeneous ensemble classification based on segmented patterns. Appl. Sci. **10**(5), 1745 (2020). https://doi.org/10.3390/app10051745
59. Yao, J., Shepperd, M.: The impact of using biased performance metrics on software defect prediction research. Inf. Softw. Technol. **139**, 106664 (2021). https://doi.org/10.1016/j.infsof.2021.106664
60. Yu, X., Keung, J., Xiao, Y., Feng, S., Li, F., Dai, H.: Predicting the precise number of software defects: are we there yet? Inf. Softw. Technol. **146**, 106847 (2022). https://doi.org/10.1016/j.infsof.2022.106847
61. Batool, I., Khan, T.A.: Software fault prediction using data mining, machine learning and deep learning techniques: a systematic literature review. Comput. Electr. Eng. **100**, 107886 (2022). https://doi.org/10.1016/j.compeleceng.2022.107886
62. Wu, X., Zheng, W., Chen, X., Zhao, Y., Yu, T., Mu, D.: Improving high-impact bug report prediction with combination of interactive machine learning and active learning. Inf. Softw. Technol. **133**, 106530 (2021). https://doi.org/10.1016/j.infsof.2021.106530
63. Giray, G.: A software engineering perspective on engineering machine learning systems: state of the art and challenges. J. Syst. Softw. **180**, 111031 (2021). https://doi.org/10.1016/j.jss.2021.111031
64. Houssein, E.H., Saeed, M.K., Al-Sayed, M.M.: Ewso: boosting white shark optimizer for solving engineering design and combinatorial problems. Math. Comput. Simul. (2023). https://doi.org/10.1016/j.matcom.2023.11.019
65. Braik, M., Hammouri, A., Atwan, J., Al-Betar, M.A., Awadallah, M.A.: White shark optimizer: a novel bio-inspired meta-heuristic algorithm for global optimization problems. Knowl. Based Syst. **243**, 108457 (2022). https://doi.org/10.1016/j.knosys.2022.108457
66. Chantar, H., Mafarja, M., Alsawalqah, H., Heidari, A.A., Aljarah, I., Faris, H.: Feature selection using binary grey wolf optimizer with elite-based crossover for Arabic text classification. Neural Comput. Appl. **32**, 12201–12220 (2020). https://doi.org/10.1007/s00521-019-04368-6
67. Ali, M.A., Kamel, S., Hassan, M.H., Ahmed, E.M., Alanazi, M.: Optimal power flow solution of power systems with renewable energy sources using white sharks algorithm. Sustainability **14**(10), 6049 (2022). https://doi.org/10.3390/su14106049
68. Makhadmeh, S.N., Al-Betar, M.A., Assaleh, K., Kassaymeh, S.: A hybrid white shark equilibrium optimizer for power scheduling problem based iot. IEEE Access **10**, 132212–132231 (2022). https://doi.org/10.1109/ACCESS.2022.3229434
69. Zhang, R., Li, X., Ding, Y., Ren, H.: Uav path planning method based on modified white shark optimization. In: 2022 IEEE International Conference on Unmanned Systems (ICUS), pp. 380–386. IEEE (2022). https://doi.org/10.1109/ICUS55513.2022.9987109
70. Liang, J., Liu, L.: Optimal path planning method for unmanned surface vehicles based on improved shark-inspired algorithm. J. Mar. Sci. Eng. **11**(7), 1386 (2023). https://doi.org/10.3390/jmse11071386
71. Fathy, A., Yousri, D., Alharbi, A.G., Abdelkareem, M.A.: A new hybrid white shark and whale optimization approach for estimating the li-ion battery model parameters. Sustainability **15**(7), 5667 (2023). https://doi.org/10.3390/su15075667
72. Amor, N., Noman, M.T., Petru, M., Sebastian, N., Balram, D.: Design and optimization of machinability of zno embedded-glass fiber reinforced polymer composites with a modified white shark optimizer. Expert Syst. Appl. **237**, 121474 (2024). https://doi.org/10.1016/j.eswa.2023.121474
73. Supreeth, S., Bhargavi, S., Margam, R., Annaiah, H., Nandalike, R.: Virtual machine placement using Adam white shark optimization algorithm in cloud computing. SN Comput. Sci. **5**(1), 21 (2023). https://doi.org/10.1007/s42979-023-02341-8
74. Lakshmanan, M., Kumar, C., Jasper, J.S.: Optimal parameter characterization of an enhanced mathematical model of solar photovoltaic cell/module using an improved white shark optimization algorithm. Optim. Control Appl. Methods (2023). https://doi.org/10.1002/oca.2984
75. Khleel, N.A.A., Nehez, K.: A novel approach for software defect prediction using cnn and gru based on smote tomek method. J. Intell. Inf. Syst. **60**(3), 673–707 (2023). https://doi.org/10.1007/s10844-023-00793-1
76. Rathore, S.S., Kumar, S.: An empirical study of ensemble techniques for software fault prediction. Appl. Intell. **51**, 3615–3644 (2021). https://doi.org/10.1007/s10489-020-01935-6
77. Wang, K., Liu, L., Yuan, C., Wang, Z.: Software defect prediction model based on lasso-svm. Neural Comput. Appl. **33**, 8249–8259 (2021). https://doi.org/10.1007/s00521-020-04960-1
78. Goyal, S.: Predicting the defects using stacked ensemble learner with filtered dataset. Autom. Softw. Eng. **28**(2), 14 (2021). https://doi.org/10.1007/s10515-021-00285-y
79. Ali, U., Aftab, S., Iqbal, A., Nawaz, Z., Bashir, M.S., Saeed, M.A.: Software defect prediction using variant based ensemble learning and feature selection techniques. Int. J. Mod. Educ. Comput. Sci **12**(5), 29–40 (2020). https://doi.org/10.5815/ijmecs.2020.05.03
80. Suresh Kumar, P., Behera, H.S., Nayak, J., Naik, B.: Bootstrap aggregation ensemble learning-based reliable approach for software defect prediction by using characterized code feature. Innov. Syst. Softw. Eng. **17**(4), 355–379 (2021). https://doi.org/10.1007/s11334-021-00399-2
81. Alazba, A., Aljamaan, H.: Software defect prediction using stacking generalization of optimized tree-based ensembles. Appl. Sci. **12**(9), 4577 (2022). https://doi.org/10.3390/app12094577
82. Jacob, R.J., Kamat, R.J., Sahithya, N., John, S.S., Shankar, S.P.: Voting based ensemble classification for software defect prediction. In: 2021 IEEE Mysore Sub Section International Conference (MysuruCon), pp. 358–365. IEEE (2021). https://doi.org/10.1109/MysuruCon52639.2021.9641713
83. Bhutamapuram, U.S., Sadam, R.: With-in-project defect prediction using bootstrap aggregation based diverse ensemble learning technique. J. King Saud Univ. Comput. Inf. Sci. **34**(10), 8675–8691 (2022). https://doi.org/10.1016/j.jksuci.2021.09.010
84. Mangla, M., Sharma, N., Mohanty, S.N.: A sequential ensemble model for software fault prediction. Innov. Syst. Softw. Eng. **18**(2), 301–308 (2022). https://doi.org/10.1007/s11334-021-00390-x
85. Zheng, J., Wang, X., Wei, D., Chen, B., Shao, Y.: A novel imbalanced ensemble learning in software defect predication. IEEE Access **9**, 86855–86868 (2021). https://doi.org/10.1109/ACCESS.2021.3072682
86. Alsaedi, S.A., Noaman, A.Y., Gad-Elrab, A.A., Eassa, F.E.: Nature-based prediction model of bug reports based on ensemble machine learning model. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3288156

87. Chen, J., Xu, J., Cai, S., Wang, X., Chen, H., Li, Z.: Software defect prediction approach based on a diversity ensemble combined with neural network. IEEE Trans. Reliab. (2024). https://doi.org/10.1109/TR.2024.3356515
88. Ali, M., Mazhar, T., Arif, Y., Al-Otaibi, S., Ghadi, Y.Y., Shahzad, T., Khan, M.A., Hamam, H.: Software defect prediction using an intelligent ensemble-based model. IEEE Access (2024). https://doi.org/10.1109/ACCESS.2024.3358201
89. Panda, R.R., Nagwani, N.K.: Software bug severity and priority prediction using smote and intuitionistic fuzzy similarity measure. Appl. Soft Comput. **150**, 111048 (2024). https://doi.org/10.1016/j.asoc.2023.111048
90. Ceran, A.A., Ar, Y., Tanriover, O., Ceran, S.S.: Prediction of software quality with machine learning-based ensemble methods. Mater. Today: Proc. **81**, 18–25 (2023). https://doi.org/10.1016/j.matpr.2022.11.229
91. Johnson, F., Oluwatobi, O., Folorunso, O., Ojumu, A.V., Quadri, A.: Optimized ensemble machine learning model for software bugs prediction. Innov. Syst. Softw. Eng. **19**(1), 91–101 (2023). https://doi.org/10.1007/s11334-022-00506-x
92. Bansal, K., Singh, G., Malik, S., Rohil, H.: Nrpredictor: an ensemble learning and feature selection based approach for predicting the non-reproducible bugs. Int. J. Syst. Assur. Eng. Manag. **14**(3), 989–1009 (2023). https://doi.org/10.1007/s13198-023-01902-7
93. Jiang, F., Yu, X., Gong, D., Du, J.: A random approximate reduct-based ensemble learning approach and its application in software defect prediction. Inf. Sci. **609**, 1147–1168 (2022). https://doi.org/10.1016/j.ins.2022.07.130
94. Chen, L., Wang, C., Song, S.-L.: Software defect prediction based on nested-stacking and heterogeneous feature selection. Complex Intell. Syst. **8**(4), 3333–3348 (2022). https://doi.org/10.1007/s40747-022-00676-y
95. Houssein, E.H., Oliva, D., Celik, E., Emam, M.M., Ghoniem, R.M.: Boosted sooty tern optimization algorithm for global optimization and feature selection. Expert Syst. Appl. **213**, 119015 (2023). https://doi.org/10.1016/j.eswa.2022.119015
96. Dokeroglu, T., Deniz, A., Kiziloz, H.E.: A comprehensive survey on recent metaheuristics for feature selection. Neurocomputing **494**, 269–296 (2022). https://doi.org/10.1016/j.neucom.2022.04.083
97. Mafarja, M., Thaher, T., Too, J., Chantar, H., Turabieh, H., Houssein, E.H., Emam, M.M.: An efficient high-dimensional feature selection approach driven by enhanced multi-strategy grey wolf optimizer for biological data classification. Neural Comput. Appl. **35**(2), 1749–1775 (2023). https://doi.org/10.1007/s00521-022-07836-8
98. Houssein, E.H., Emam, M.M., Ali, A.A.: Improved manta ray foraging optimization for multi-level thresholding using covid-19 ct images. Neural Comput. Appl. **33**(24), 16899–16919 (2021). https://doi.org/10.1007/s00521-021-06273-3
99. Raslan, M.A., Raslan, S.A., Shehata, E.M., Mahmoud, A.S., Sabri, N.A.: Advances in the applications of bioinformatics and chemoinformatics. Pharmaceuticals **16**(7), 1050 (2023). https://doi.org/10.3390/ph16071050
100. Tougaccar, M.: Disease type detection in lung and colon cancer images using the complement approach of inefficient sets. Comput. Biol. Med. **137**, 104827 (2021). https://doi.org/10.1016/j.compbiomed.2021.104827
101. Zivkovic, T., Nikolic, B., Simic, V., Pamucar, D., Bacanin, N.: Software defects prediction by metaheuristics tuned extreme gradient boosting and analysis based on Shapley additive explanations. Appl. Soft Comput. **146**, 110659 (2023). https://doi.org/10.1016/j.asoc.2023.110659
102. Prashanthi, M., Chandra Mohan, M.: Hybrid optimization-based neural network classifier for software defect prediction. Int. J. Image Graph. **24**(04), 2450045 (2024). https://doi.org/10.1142/S0219467824500451
103. Goyal, S., Bhatia, P.K.: Software fault prediction using lion optimization algorithm. Int. J. Inf. Technol. **13**, 2185–2190 (2021). https://doi.org/10.1007/s41870-021-00804-w
104. Shafiq, M., Alghamedy, F.H., Jamal, N., Kamal, T., Daradkeh, Y.I., Shabaz, M.: Retracted: scientific programming using optimized machine learning techniques for software fault prediction to improve software quality. IET Softw. **17**(4), 694–704 (2023). https://doi.org/10.1049/sfw2.12091
105. Das, H., Prajapati, S., Gourisaria, M.K., Pattanayak, R.M., Alameen, A., Kolhar, M.: Feature selection using golden jackal optimization for software fault prediction. Mathematics **11**(11), 2438 (2023). https://doi.org/10.3390/math11112438
106. Thaher, T., Arman, N.: Efficient multi-swarm binary harris hawks optimization as a feature selection approach for software fault prediction. In: 2020 11th International Conference on Information and Communication Systems (ICICS), pp. 249–254. IEEE (2020). https://doi.org/10.1109/ICICS49469.2020.239557
107. Alawad, N.A., Abed-alguni, B.H., Al-Betar, M.A., Jaradat, A.: Binary improved white shark algorithm for intrusion detection systems. Neural Comput. Appl. (2023). https://doi.org/10.1007/s00521-023-08772-x
108. Fathy, A., Alanazi, A.: An efficient white shark optimizer for enhancing the performance of proton exchange membrane fuel cells. Sustainability **15**(15), 11741 (2023). https://doi.org/10.3390/su151511741
109. Saadi, A.A., Soukane, A., Meraihi, Y., Gabis, A.B., Ramdane-Cherif, A., Yahia, S.: An enhanced white shark optimization algorithm for unmanned aerial vehicles placement. In: EAI International Conference on Computational Intelligence and Communications, pp. 27–42. Springer (2022). https://doi.org/10.1007/978-3-031-34459-6_3
110. Taleb, S.M., Meraihi, Y., Ramdane-Cherif, A., Gabis, A.B., Acheli, D.: Enhanced white shark optimization algorithm for the mesh routers placement problem with service priority in wireless mesh networks. In: Workshop on Mining Data for Financial Applications, pp. 183–196. Springer (2022).https://doi.org/10.1007/978-981-99-1620-7_15
111. Wu, N., Zhang, G., Fan, S., Huang, Y.: Optimal scheduling of grid connected microgrid based on improved white shark algorithm. In: 2022 China Automation Congress (CAC), pp. 5712–5717. IEEE (2022).https://doi.org/10.1109/CAC57257.2022.10055618
112. Li, Y., Tang, B., Huang, B., Xue, X.: A dual-optimization fault diagnosis method for rolling bearings based on hierarchical slope entropy and svm synergized with shark optimization algorithm. Sensors **23**(12), 5630 (2023). https://doi.org/10.3390/s23125630
113. Chandok, G.A., Rexy, V., Basha, H.A., Selvi, H.: Enhancing bankruptcy prediction with white shark optimizer and deep learning: A hybrid approach for accurate financial risk assessment. Int. J. Intell. Eng. Syst. (2024). https://doi.org/10.22266/ijies2024.0229.14
114. Durga, N., Gayathri, T., Kumari, K.R., Madhavi, T.: Clustering based hybrid optimized model for effective data transmission. In: International Conference on Cognitive Computing and Cyber Physical Systems, pp. 338–351. Springer (2023). https://doi.org/10.1007/978-3-031-48891-7_30
115. Parveen, N., Chakrabarti, P., Hung, B.T., Shaik, A.: Twitter sentiment analysis using hybrid gated attention recurrent network. J. Big Data **10**(1), 1–29 (2023). https://doi.org/10.1186/s40537-023-00726-3
116. Xing, Q., Wang, J., Jiang, H., Wang, K.: Research of a novel combined deterministic and probabilistic forecasting system for air pollutant concentration. Expert Syst. Appl. **228**, 120117 (2023). https://doi.org/10.1016/j.eswa.2023.120117

117. Kumaresan, K., Rohith Bhat, C., Lalitha Devi, K.: A novel fuzzy marine white shark optimization based efficient routing and enhancing network lifetime in manet. Wirel. Pers. Commun. **132**(4), 2363–2385 (2023). https://doi.org/10.1007/s11277-023-10675-y
118. Sahu, V.S.D.M., Samal, P., Panigrahi, C.K.: Tyrannosaurus optimization algorithm: a new nature-inspired meta-heuristic algorithm for solving optimal control problems. e-Prime-Adv. Electr. Eng. Electron. Energy **5**, 100243 (2023). https://doi.org/10.1016/j.prime.2023.100243
119. Chandana Mani, R., Kamalakannan, J.: Computer-aided diagnosis using white shark optimizer with attention-based deep learning for breast cancer classification. J. Intell. Fuzzy Syst. (2023). https://doi.org/10.3233/JIFS-231776
120. Alwayle, I.M., Al-onazi, B.B., Nour, M.K., Alalayah, K.M., Alaidarous, K.M., Ahmed, I.A., Mehanna, A.S., Motwakel, A.: Automated spam review detection using hybrid deep learning on Arabic opinions. Comput. Syst. Sci. Eng. (2023). https://doi.org/10.32604/csse.2023.034456
121. Gude, M.K., Salma, U.: A novel white shark optimizer for optimal parameter selection of power system oscillation damper. In: Soft Computing Applications in Modern Power and Energy Systems: Select Proceedings of EPREC 2022, pp. 217–226. Springer, Singapore (2023). https://doi.org/10.1007/978-981-19-8353-5_15
122. Kalaivani, C., Umasankar, L., Badachi, C., Karthikumar, K.: An enhanced approach-based grid flexibility analysis for combined heat and power systems with variable renewable energy systems. Energy Environ. (2023). https://doi.org/10.1177/0958305X231153970
123. Ali, E.S., Abd Elazim, S.M., Hakmi, S.H., Mosaad, M.I.: Optimal allocation and size of renewable energy sources as distributed generations using shark optimization algorithm in radial distribution systems. Energies **16**(10), 3983 (2023)
124. Ali, M.A., Kamel, S., Hassan, M.H., Ahmed, E.M., Alanazi, M.: Optimal power flow solution of power systems with renewable energy sources using white sharks algorithm. Sustainability (2022). https://doi.org/10.3390/su14106049
125. Fathy, A., Alanazi, A.: An efficient white shark optimizer for enhancing the performance of proton exchange membrane fuel cells. Sustainability (2023). https://doi.org/10.3390/su151511741
126. Amor, N., Tayyab Noman, M., Petru, M., Sebastian, N., Balram, D.: Design and optimization of machinability of zno embedded-glass fiber reinforced polymer composites with a modified white shark optimizer. Expert Syst. Appl. **237**, 121474 (2024). https://doi.org/10.1016/j.eswa.2023.121474
127. Kumar, S., Sharma, N.K., Kumar, N.: Wsomark: an adaptive dual-purpose color image watermarking using white shark optimizer and Levenberg–Marquardt bpnn. Expert Syst. Appl. **226**, 120137 (2023). https://doi.org/10.1016/j.eswa.2023.120137
128. Althobaiti, T., Sanjalawe, Y., Ramzan, N.: Securing cloud computing from flash crowd attack using ensemble intrusion detection system. Comput. Syst. Sci. Eng. (2023). https://doi.org/10.32604/csse.2023.039207
129. Daneshfar, F., Aghajani, M.J.: Enhanced text classification through an improved discrete laying chicken algorithm. Expert Syst. (2024). https://doi.org/10.1111/exsy.13553
130. Jafari, S., Aghaee-Maybodi, N.: Detection of phishing addresses and pages with a data set balancing approach by generative adversarial network (gan) and convolutional neural network (cnn) optimized with swarm intelligence. Concurr. Comput. Pract. Exp. (2024). https://doi.org/10.1002/cpe.8033
131. Pal, S., Sillitti, A.: Cross-project defect prediction: a literature review. IEEE Access (2022). https://doi.org/10.1109/ACCESS.2022.3221184
132. Nevendra, M., Singh, P.: A survey of software defect prediction based on deep learning. Arch. Comput. Methods Eng. **29**(7), 5723–5748 (2022). https://doi.org/10.1007/s11831-022-09787-8
133. Siddiqui, T., Mustaqeem, M.: Performance evaluation of software defect prediction with nasa dataset using machine learning techniques. Int. J. Inf. Technol. **15**(8), 4131–4139 (2023). https://doi.org/10.1007/s41870-023-01528-9
134. Mehmood, I., Shahid, S., Hussain, H., Khan, I., Ahmad, S., Rahman, S., Ullah, N., Huda, S.: A novel approach to improve software defect prediction accuracy using machine learning. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3287326
135. Dong, X., Liang, Y., Miyamoto, S., Yamaguchi, S.: Ensemble learning based software defect prediction. J. Eng. Res. **11**(4), 377–391 (2023). https://doi.org/10.1016/j.jer.2023.10.038
136. Mostefai, A.: Using sum product networks to predict defects in software systems. Int. J. Inf. Technol. (2024). https://doi.org/10.1007/s41870-024-02067-7
137. Wongvorachan, T., He, S., Bulut, O.: A comparison of undersampling, oversampling, and smote methods for dealing with imbalanced classification in educational data mining. Information **14**(1), 54 (2023). https://doi.org/10.3390/info14010054
138. Ciubotariu, G., Czibula, G., Czibula, I.G., Chelaru, I.-G.: Uncovering behavioural patterns of one: and binary-class svm-based software defect predictors. In: ICSOFT, pp. 249–257. SCITEPRESS – Science and Technology Publications (2023). https://doi.org/10.5220/0012052700003538
139. Cai, X., Niu, Y., Geng, S., Zhang, J., Cui, Z., Li, J., Chen, J.: An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search. Concurr. Comput. Pract. Exp. **32**(5), 5478 (2020). https://doi.org/10.1002/cpe.5478
140. Sun, Z., Wang, G., Li, P., Wang, H., Zhang, M., Liang, X.: An improved random forest based on the classification accuracy and correlation measurement of decision trees. Expert Syst. Appl. **237**, 121549 (2024). https://doi.org/10.1016/j.eswa.2023.121549
141. Wang, F., Ma, S., Wang, H., Li, Y., Qin, Z., Zhang, J.: A hybrid model integrating improved flower pollination algorithm-based feature selection and improved random forest for nox emission estimation of coal-fired power plants. Measurement **125**, 303–312 (2018). https://doi.org/10.1016/j.measurement.2018.04.069
142. Sharma, T., Jatain, A., Bhaskar, S., Pabreja, K.: Ensemble machine learning paradigms in software defect prediction. Procedia Comput. Sci. **218**, 199–209 (2023). https://doi.org/10.1016/j.procs.2023.01.002
143. Stradowski, S., Madeyski, L.: Machine learning in software defect prediction: a business-driven systematic mapping study. Inf. Softw. Technol. **155**, 107128 (2023). https://doi.org/10.1016/j.infsof.2022.107128
144. Rodriguez Sanchez, E., Vazquez Santacruz, E.F., Cervantes Maceda, H.: Effort and cost estimation using decision tree techniques and story points in agile software development. Mathematics **11**(6), 1477 (2023). https://doi.org/10.3390/math11061477
145. Goyal, S.: 3pcge: 3-parent child-based genetic evolution for software defect prediction. Innov. Syst. Softw. Eng. **19**(2), 197–216 (2023). https://doi.org/10.1007/s11334-021-00427-1
146. Zhao, Y., Damevski, K., Chen, H.: A systematic survey of just-in-time software defect prediction. ACM Comput. Surv. **55**(10), 1–35 (2023). https://doi.org/10.1145/3567550
147. Feyzi, F., Daneshdoost, A.: Studying the effectiveness of deep active learning in software defect prediction. Int. J. Comput. Appl. **45**(7–8), 534–552 (2023). https://doi.org/10.1080/1206212X.2023.2252117
148. Babatunde, A.N., Ogundokun, R.O., Adeoye, L.B., Misra, S.: Software defect prediction using dagging meta-learner-based classifiers. Mathematics **11**(12), 2714 (2023). https://doi.org/10.3390/math11122714

149. Shehab, M.A., Khreich, W., Hamou-Lhadj, A., Sedki, I.: Commit-time defect prediction using one-class classification. J. Syst. Softw. **208**, 111914 (2024). https://doi.org/10.1016/j.jss.2023.111914
150. Oleshchenko, L.: Software testing errors classification method using clustering algorithms. In: International Conference On Innovative Computing And Communication, pp. 553–566. Springer (2023). https://doi.org/10.1007/978-981-99-3315-0_42
151. Banga, M., Bansal, A.: Proposed software faults detection using hybrid approach. Secur. Priv. **6**(4), 103 (2023). https://doi.org/10.1002/spy2.103
152. Tahir, T., Gencel, C., Rasool, G., Tariq, U., Rasheed, J., Yeo, S.F., Cevik, T.: Early software defects density prediction: training the international software benchmarking cross projects data using supervised learning. IEEE Access (2023). https://doi.org/10.1109/ACCESS.2023.3339994
153. Dinter, R., Catal, C., Giray, G., Tekinerdogan, B.: Just-in-time defect prediction for mobile applications: using shallow or deep learning? Softw. Qual. J. **31**(4), 1281–1302 (2023). https://doi.org/10.1007/s11219-023-09629-1
154. Dewangan, S., Rao, R.S., Chowdhuri, S.R., Gupta, M.: Severity classification of code smells using machine-learning methods. SN Comput. Sci. **4**(5), 564 (2023). https://doi.org/10.1007/s42979-023-01979-8
155. Pandey, S., Kumar, K.: Software fault prediction for imbalanced data: a survey on recent developments. Procedia Comput. Sci. **218**, 1815–1824 (2023). https://doi.org/10.1016/j.procs.2023.01.159
156. Douiba, M., Benkirane, S., Guezzaz, A., Azrour, M.: An improved anomaly detection model for iot security using decision tree and gradient boosting. J. Supercomput. **79**(3), 3392–3411 (2023). https://doi.org/10.1007/s11227-022-04783-y
157. Li, L., Su, R., Zhao, X.: Neighbor cleaning learning based cost-sensitive ensemble learning approach for software defect prediction. Concurr. Comput. Pract. Exp. **36**(12), 8017 (2024). https://doi.org/10.1002/cpe.8017
158. Tao, H., Niu, X., Xu, L., Fu, L., Cao, Q., Chen, H., Shang, S., Xian, Y.: A comparative study of software defect binomial classification prediction models based on machine learning. Softw. Qual. J. (2024). https://doi.org/10.1007/s11219-024-09683-3
159. Ampomah, E.K., Qin, Z., Nyame, G.: Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement. Information **11**(6), 332 (2020). https://doi.org/10.3390/info11060332
160. Mcmurray, S., Sodhro, A.H.: A study on ml-based software defect detection for security traceability in smart healthcare applications. Sensors **23**(7), 3470 (2023). https://doi.org/10.3390/s23073470
161. Nascimento, L.P.G., Prudencio, R.B.C., Mota, A.C., Filho, A.d.A.P., Cruz, P.H.A., Oliveira, D.C.C.A.d., Moreira, P.R.S.: Machine learning techniques for escaped defect analysis in software testing. In: Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing, pp. 47–53. ACM Digital Library (2023). https://doi.org/10.1145/3624032.3624039
162. Al-Isawi, M.K., Abdulkader, H.: Software defects detection in explainable machine learning approach. In: International Conference on Emerging Trends and Applications in Artificial Intelligence, pp. 505–519. Springer (2023). https://doi.org/10.1007/978-3-031-56728-5_42
163. Azzeh, M., Nassif, A.B., Talib, M.A., Iqba, H.: Software defect prediction using non-dominated sorting genetic algorithm and k-nearest neighbour classifier. e-Inform. Softw. Eng. J. (2024). https://doi.org/10.37190/e-Inf240103
164. Wolff, M.K., Schaathun, H.G., Fougner, A.L., Steinert, M., Volden, R.: Mobile software development kit for real time multivariate blood glucose prediction. IEEE Access (2024). https://doi.org/10.1109/ACCESS.2024.3349496
165. Barandier, P., Mendes, M., Cardoso, A.J.M.: Comparative analysis of four classification algorithms for fault detection of heat pumps. Energy Build (2024). https://doi.org/10.1016/j.enbuild.2024.114342
166. Khatri, Y., Singh, S.K.: Cross project defect prediction: a comprehensive survey with its swot analysis. Innov. Syst. Softw. Eng. (2022). https://doi.org/10.1007/s11334-020-00380-5
167. Nhu, V.-H., Shirzadi, A., Shahabi, H., Singh, S.K., Al-Ansari, N., Clague, J.J., Jaafari, A., Chen, W., Miraki, S., Dou, J., et al.: Shallow landslide susceptibility mapping: A comparison between logistic model tree, logistic regression, naive bayes tree, artificial neural network, and support vector machine algorithms. Int. J. Environ. Res. Public Health **17**(8), 2749 (2020). https://doi.org/10.3390/ijerph17082749
168. Lopez-Martin, C., Villuendas-Rey, Y., Azzeh, M., Nassif, A.B., Banitaan, S.: Transformed k-nearest neighborhood output distance minimization for predicting the defect density of software projects. J. Syst. Softw. **167**, 110592 (2020). https://doi.org/10.1016/j.jss.2020.110592
169. Mustaqeem, M., Mustajab, S., Alam, M.: A hybrid approach for optimizing software defect prediction using a grey wolf optimization and multilayer perceptron. Int. J. Intell. Comput. Cybern. **17**(2), 436–464 (2024). https://doi.org/10.1108/IJICC-11-2023-0385
170. Kassaymeh, S., Al-Betar, M.A., Rjoubd, G., Fraihat, S., Abdullah, S., Almasri, A.: Optimizing beyond boundaries: empowering the salp swarm algorithm for global optimization and defective software module classification. Neural Comput. Appl. (2024). https://doi.org/10.1007/s00521-024-10131-3
171. Aleem, S., Capretz, L.F., Ahmed, F.: Benchmarking machine learning technologies for software defect detection (2015). https://doi.org/10.48550/arXiv.1506.07563
172. Iqbal, A., Aftab, S.: A classification framework for software defect prediction using multi-filter feature selection technique and mlp. Int. J. Mod. Educ. Comput. Sci. (2020). https://doi.org/10.5815/ijmecs.2020.01.03
173. Anand, K., Jena, A.K., et al.: Forecasting software modules with known defects through the quadratic discriminant analysis feature reduction technique. In: 2024 International Conference on Emerging Systems and Intelligent Computing (ESIC), pp. 713–717. IEEE (2024). https://doi.org/10.1109/ESIC60604.2024.10481542
174. Brobbey, A., Wiebe, S., Nettel-Aguirre, A., Josephson, C.B., Williamson, T., Lix, L.M., Sajobi, T.T.: Repeated measures discriminant analysis using multivariate generalized estimation equations. Stat. Methods Med. Res. **31**(4), 646–657 (2022). https://doi.org/10.1177/09622802211032705
175. Abdulhafedh, A.: Comparison between common statistical modeling techniques used in research, including: discriminant analysis vs logistic regression, ridge regression vs lasso, and decision tree vs random forest. Open Access Libr. J. **9**(2), 1–19 (2022). https://doi.org/10.4236/oalib.1108414
176. Wang, Y., Sun, P.: A fault diagnosis methodology for nuclear power plants based on kernel principle component analysis and quadratic support vector machine. Ann. Nucl. Energy **181**, 109560 (2023). https://doi.org/10.1016/j.anucene.2022.109560
177. Yang, C., Ma, S., Han, Q.: Robust discriminant latent variable manifold learning for rotating machinery fault diagnosis. Eng. Appl. Artif. Intell. **126**, 106996 (2023). https://doi.org/10.1016/j.engappai.2023.106996
178. Lin, S., Zheng, H., Han, B., Li, Y., Han, C., Li, W.: Comparative performance of eight ensemble learning approaches for the development of models of slope stability prediction. Acta Geotech.

17(4), 1477–1502 (2022). https://doi.org/10.1007/s11440-021-01440-1
179. Samantaray, R., Das, H.: Performance analysis of machine learning algorithms using bagging ensemble technique for software fault prediction. In: 2023 6th International Conference on Information Systems and Computer Networks (ISCON), pp. 1–7. IEEE (2023). https://doi.org/10.1109/ISCON57294.2023.10111952
180. Gupta, N., Sinha, R.R., Goyal, A., Sunda, N., Sharma, D.: Analyze the performance of software by machine learning methods for fault prediction techniques. IJRITCC **11**(5s), 178187 (2023). https://doi.org/10.17762/ijritcc.v11i5s.6642
181. Kulkarni, S.P., Patel, S.: Ensemble-based software fault prediction with two staged data pre-processing. Int. J. Comput. Appl. Technol. **72**(3), 212–222 (2023). https://doi.org/10.1504/IJCAT.2023.133297
182. Gupta, M., Rajnish, K., Bhattacharya, V.: Effectiveness of ensemble classifier over state-of-art machine learning classifiers for predicting software faults in software modules. In: Machine Learning, Image Processing, Network Security and Data Sciences: Select Proceedings of 3rd International Conference on MIND 2021, pp. 77–88. Springer (2023). https://doi.org/10.1007/978-981-19-5868-7_7
183. Olivares-Galindo, J.A., Sanchez-Garcia, A.J., Barrientos-Martinez, R.E., Ocharan-Hernandez, J.O.: Ensemble classifiers in software defect prediction: a systematic literature review. In: 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT), pp. 1–8. IEEE (2023).https://doi.org/10.1109/CONISOFT58849.2023.00011
184. Rajbongshi, A., Shakil, R., Akter, B., Lata, M.A., Joarder, M.M.A.: A comprehensive analysis of feature ranking-based fish disease recognition. Array **21**, 100329 (2024). https://doi.org/10.1016/j.array.2023.100329
185. Curebal, F., Dag, H.: Enhancing malware classification: A comparative study of feature selection models with parameter optimization. In: 2024 Systems and Information Engineering Design Symposium (SIEDS), pp. 511–516. IEEE (2024). https://doi.org/10.1109/SIEDS61124.2024.10534669
186. Borandag, E.: Software fault prediction using an rnn-based deep learning approach and ensemble machine learning techniques. Appl. Sci. **13**(3), 1639 (2023). https://doi.org/10.3390/app13031639
187. Singh, M., Chhabra, J.K.: Improved software fault prediction using new code metrics and machine learning algorithms. J. Comput. Lang. **78**, 101253 (2024). https://doi.org/10.1016/j.cola.2023.101253
188. Oleshchenko, L.: Machine learning algorithms comparison for software testing errors classification automation. In: International Conference on Computer Science, Engineering and Education Applications, pp. 615–625. Springer (2023). https://doi.org/10.1007/978-3-031-36118-0_55
189. Mao, R., Zhang, L., Zhang, X.: Mutation-based data augmentation for software defect prediction. J. Softw. Evol. Process (2024). https://doi.org/10.1002/smr.2634