Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics e-ISSN: 2656-8624
Homepage: https://ijeeemi.org/; Vol. 8, No. 3, pp. 248-261, May 2026

RESEARCH PAPER **OPEN ACCES**

Metaheuristic-Based Hyperparameter Optimization Analysis of Deep Neural Network for Cross-Project Defect Prediction in Mobile Applications

Maulana Abdul Rahman[1], Rudy Herteno[2], Radityo Adi Nugroho[3] , Friska Abadi[4] , and Setyo Wahyu Saputro[5]

Department of Computer Science, Faculty of Mathematics and Natural Science, Lambung Mangkurat University, Banjarbaru, Indonesia

Abstract

Software Defect Prediction (SDP) plays a strategic role in identifying software defects during the early stages of development, thereby enabling more efficient allocation of testing resources, particularly in the rapidly evolving mobile application domain characterized by fast release cycles. The commonly used Within-Project Defect Prediction (WPDP) approach is often constrained by the limited availability of historical data, especially in projects at early stages of development. As an alternative, Cross-Project Defect Prediction (CPDP) leverages historical data from other projects as training sources. Moreover, the performance of the Deep Neural Network (DNN) used in SDP is highly dependent on accurate hyperparameter configurations, where manual tuning requires substantial time and computational resources without guaranteeing optimal results. To address this issue, this study analyzes and compares the effectiveness of three metaheuristic algorithms, namely Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Grey Wolf Optimizer (GWO), in optimizing DNN hyperparameters within a CPDP framework. This study utilizes 14 open-source Android mobile application projects and employs the Leave-One-Out Cross-Validation technique. The performance of each combination is evaluated using ROC-AUC as the primary metric. The Wilcoxon Signed-Rank Test with a Bonferroni correction is used to assess the statistical significance of the observed performance differences. The experimental results demonstrate that GWO-DNN achieves the best performance, with an average ROC-AUC of 0.721, and is the only combination that remains statistically significant after Bonferroni correction, with a small effect size based on Cliff's delta. Overall, the findings of this study indicate that metaheuristic-based hyperparameter tuning is a sufficiently effective approach for improving the capability of DNN in cross-project software defect prediction within the mobile application domain, although the observed improvements remain moderate.

Paper History

Received April 22, 2026
Revised May 10, 2026
Accepted May 12, 2026
Published May 18, 2026

Keywords

Metaheuristic Algorithm;
Deep Neural Network;
Hyperparameter Tuning;
Cross-Project Defect Prediction;
Mobile Applications;

Author Email

2211016110012@mhs.ulm.ac.id
rudy.herteno@ulm.ac.id
radityo.adi@ulm.ac.id
friska.abadi@ulm.ac.id
setyo.saputro@ulm.ac.id

I. Introduction

Advancements in digitalization have established software as an essential component across various sectors of modern life, including defense, aerospace, manufacturing, financial services, and the energy industry [1]. As the complexity of software systems increases, software defects become an increasingly unavoidable issue and may lead to significant financial losses for organizations [2], [3]. The process of identifying software defects often requires substantial time and resources [4]. Therefore, Software Defect Prediction (SDP) plays a strategic role in identifying potential defects at early stages of development, thereby enabling more efficient allocation of testing resources [5], [6]. Most SDP methods use historical data from the same project via the Within-Project Defect Prediction (WPDP) approach [7]. However, its implementation is often constrained by the limited availability of historical data in newly developed projects [8]. As an alternative, Cross-Project Defect Prediction (CPDP) leverages defect data from other projects as training data to identify software modules likely to contain defects [9].

Deep Learning (DL)-based approaches have demonstrated superior performance compared to conventional Machine Learning (ML) methods in Software Defect Prediction (SDP) due to their capability to automatically extract features and learn complex data patterns [10], [11]. Among the various existing architectures, the Deep Neural Network (DNN) is capable of effectively modeling complex patterns derived from code metrics and commit histories for accurate software defect prediction [12]. Recent studies in Software Defect

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Prediction (SDP) have begun adopting Transformer-based approaches, such as CodeBERT, to enhance the semantic representation of source code. CodeBERT is a pre-trained model based on the Transformer architecture that is capable of learning code context through the self-attention mechanism [13]. Pan et al. [14] demonstrated that CodeBERT achieves competitive performance within a CPDP framework. However, Transformer-based models generally require substantial computational resources and memory due to the complexity of the self-attention mechanism [15]. Therefore, DNN remains a relevant approach for application in CPDP scenarios. Nevertheless, the performance of a DNN is highly dependent on accurate hyperparameter configurations, where manual tuning requires considerable time and resources without guaranteeing optimal results [16], [17].

Conventional methods such as Grid Search and Random Search exhibit fundamental limitations, particularly their high computational cost when applied to complex search spaces [18]. This limitation has encouraged researchers to explore metaheuristic algorithms [19] due to their capability to navigate search spaces more efficiently and systematically [20], [21]. Mumtahina et al. [22] state that Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Grey Wolf Optimization (GWO) are among the most widely adopted metaheuristic algorithms for hyperparameter tuning. This finding is supported by Saputri et al. [23], who emphasize the superiority of these algorithms in balancing exploration and exploitation within the search space.

Although numerous studies have investigated metaheuristic-based hyperparameter tuning, several research gaps remain unresolved. First, most existing studies apply metaheuristic algorithms for hyperparameter tuning outside the SDP domain , such as the work by Saputri et al. [23] in the context of entrepreneurial fear classification. Consequently, systematic comparisons among metaheuristic algorithms within the context of SDP using a DNN remain limited. Second, existing studies are predominantly conducted within the WPDP framework, whereas the application of metaheuristic-based hyperparameter tuning in CPDP scenarios remains underexplored. Third, the majority of prior studies do not perform statistical significance testing to determine whether the observed performance improvements differ significantly from those of baseline models. The combination of metaheuristic-based hyperparameter tuning and CPDP scenarios on Android mobile application datasets, supported by rigorous statistical validation, remains highly limited. Therefore, this study addresses these gaps simultaneously within a systematic experimental framework.

To address these gaps, this study conducts a metaheuristic-based optimization analysis for DNN hyperparameter tuning within a CPDP framework using 14 open-source Android mobile application datasets. Three metaheuristic algorithms, namely PSO, GA, and GWO, are employed to automatically optimize the DNN hyperparameters. The performance of each combination is evaluated using the Receiver Operating Characteristic Area Under the Curve (ROC-AUC) as the primary evaluation metric, supported by additional metrics including F1-Score, Precision-Recall Area Under the Curve (PR-AUC), and Matthews Correlation Coefficient (MCC) to provide a more comprehensive performance assessment. All combinations are compared against a baseline DNN model without tuning and validated using the Wilcoxon Signed-Rank Test to determine the statistical significance of the observed performance differences.

This study aims to analyze and compare the effectiveness of PSO, GA, and GWO in optimizing DNN hyperparameters within a CPDP framework, evaluated using the ROC-AUC, F1-Score, PR-AUC, and MCC metrics. In addition, this study examines the statistical significance of the resulting performance improvements using the Wilcoxon Signed-Rank Test, thereby providing recommendations regarding the most optimal metaheuristic algorithm for DNN hyperparameter tuning in CPDP scenarios for mobile applications.

This study provides several important contributions to SDP. First, it investigates the impact of metaheuristic-based hyperparameter tuning on DNN performance within a CPDP framework for mobile applications. Second, it provides empirical evidence on the effectiveness of PSO, GA, and GWO in optimizing DNN hyperparameters through comprehensive evaluation using ROC-AUC, F1-Score, PR-AUC, and MCC metrics, supported by statistical validation using the Wilcoxon Signed-Rank Test. Third, this study extends the application of metaheuristic-based hyperparameter tuning from WPDP scenarios to CPDP scenarios. Fourth, it identifies the most reliable metaheuristic algorithm for DNN hyperparameter tuning within CPDP frameworks in the mobile application domain.

This paper is organized as follows. Section II presents the materials and methods, including the dataset, data preprocessing, data partitioning, DNN architecture, metaheuristic algorithms, and evaluation procedures. Section III reports the experimental results. Section IV discusses the interpretation of the results, comparisons with previous studies, limitations, and research implications. Finally, Section V concludes the study by summarizing the main findings and outlining directions for future research.

II. Materials and Method

This study presents a structured procedural workflow, as illustrated in Fig. 1. It integrates metaheuristic algorithms to optimize hyperparameters of DNN within a CPDP framework. The process begins with data collection from open-source Android mobile application datasets comprising 14 projects, each of which contains code change metrics and defect labels. This is followed by a preprocessing stage that handles missing values and normalizes the data using MinMaxScaler to scale feature values to 0-1. The processed data are then partitioned into source and target projects using the Leave-One-Out

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Cross-Validation technique, in which each validation iteration designates one project as the target and the remaining 13 as source projects, with the process repeated 14 times.

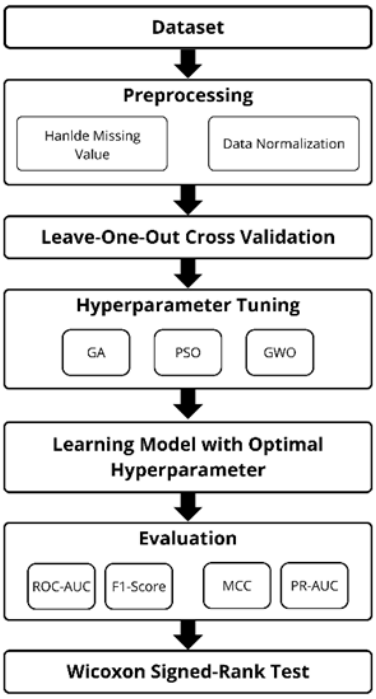

Fig. 1. Research Flow of the Proposed Study

Prior to model training, hyperparameter optimization is performed using three metaheuristic algorithms, namely GA, PSO, and GWO, to optimize the DNN, with ROC-AUC employed as the fitness function, thereby generating multiple experimental configurations. The model with the optimal hyperparameters is subsequently trained on the source project and evaluated on the target project. Model performance is assessed using ROC-AUC and compared against a baseline model without hyperparameter tuning. Furthermore, the Wilcoxon Signed-Rank Test is applied to determine whether the performance improvements achieved through hyperparameter tuning are statistically significant.

A. Dataset

This study utilizes an open-source Android mobile application dataset containing commit-level metrics along with defect labels that indicate whether a commit is likely to introduce software defects [24]. The dataset is publicly available and can be accessed via https://figshare.com/s/9a075be3e1fb64f76b48?file=14256152. In this study, 14 open-source Android mobile application projects are selected, consistent with those used by Jorayeva et al. [25], in order to capture diverse characteristics and varying levels of complexity within software defect prediction scenarios. The selected projects include Afwall, Alfresco, androidSync, androidWallpaper, anySoftKeyboard, Apg, atmosphere, chatSecure, Facebook, Flutter, Kiwis, owncloudandroid, Pageturner, Reddit. Each project differs in the numbers of commit instances, defective commits, clean commits, and defect ratios. A summary of this information is presented in Table 2. In addition, the dataset comprises six primary features that describe the characteristics of code changes for each commit, as summarized in Table 1. The dataset consists of numerical data with varying sizes across projects and comprises 30,042 commit instances. It includes six features, namely NUC, NDEV, LD, LA, NF, and ND, along with a target variable, contains_bug, which indicates whether a given commit introduces a defect [25]. These six features represent several dimensions of software metrics, specifically history, size, and diffusion, as presented in Table 1 [26]. The selection of these six features is based on the study by Catolino et al. [27], which demonstrates that these features are the most relevant predictors for commit-based defect prediction in mobile applications.

Table 1. Description of Commit-Level Metrics Used as Features

Name	Description	Dimension
NUC	Number of unique changes to modified files	History
NDEV	Number of developers working on the files	History
LD	Lines of code deleted	Size
LA	Lines of code added	Size
NF	Number of modified files	Diffusion
ND	Number of modified directories	Diffusion

Based on the data distribution in Table 2, a significant variation in defect ratios across projects is observed. Some projects, such as Flutter, exhibit very low defect ratios (1.2%), while others show relatively high defect ratios, such as Afwall (40.4%) and Atmosphere (39.7%). This condition indicates potential class imbalance, which may affect the performance of classification models. In addition, the dataset contains 36 missing values out of 30,042 instances (0.12%). The distribution of missing values across projects is relatively low and uneven, with the highest percentage observed in the Alfresco project (0.5%) and the lowest in the Flutter project (0.06%). Despite this variation, all values remain very low and therefore do not have a significant impact on the overall quality of the dataset [28].

B. Preprocessing

Data preprocessing is a critical stage that ensures the quality and consistency of raw data prior to model training [29]. In this study, two preprocessing steps are applied. The preprocessing procedure is presented in the form of pseudocode in Table 3. First, missing values are handled by removing rows containing missing entries from the dataset. This approach is considered appropriate when the proportion of missing values is relatively small, such that data removal does not significantly affect the overall representation of the dataset [30], [31]. Second, data

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Table 2. Summary of Android Mobile Apps Datasets

Dataset	Total Commit Instances	Defective Commit Instances	Clean Commit Instances	Defect Rate (%)
Afwall	1025	414	611	40.4
Alfresco	1004	214	790	21.3
androidSync	209	62	147	29.7
androidWallpaper	588	94	394	16
anySoftKeyboard	2971	819	2152	27.6
Apg	3780	1304	2476	34.5
Atmosphere	5474	2174	3300	39.7
chatSecure	2579	853	1726	33.1
Facebook	584	180	386	32.8
Flutter	10405	130	10275	1.2
kiwis	1373	350	1023	25.5
owncloudandroid	3700	830	2870	22.4
Pageturner	164	23	141	14
Reddit	222	60	162	27

normalization is performed using Min-Max Normalization to scale all feature values to the range [0, 1]. The normalization process is applied within each iteration of Leave-One-Out Cross-Validation, where the scaler is fitted only on the training data and subsequently applied to the testing data using the same parameters. This approach ensures that no information from the test data leaks into the training process, thereby preventing any feature from dominating the training process while also improving computational efficiency [32]. Eq. (1) is used in Min-Max Normalization [33].

$$x = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

Based on Eq. (1), x represents the normalized value obtained after the scaling process, while X denotes the original value before normalization. The term X_{min} refers to the minimum value of the feature in the dataset, and X_{max} indicates the maximum value of the feature [33].

C. Data Splitting

This study employs Leave-One-Out Cross-Validation technique as the data partitioning strategy within a Cross-Project Defect Prediction (CPDP) framework. In each validation iteration, one project is designated as the target project for testing, while the remaining projects are combined into a single source project for training. By applying this approach, the model's capability can be evaluated without being influenced by the specific characteristics of any individual project [8].

Table 3. Algorithm 1

Data Preprocessing	
1.	**BEGIN**
2.	FOR each dataset IN dataset_folder
3.	Load dataset
4.	Remove rows with missing values
5.	END FOR
6.	Combine all datasets
7.	Split data into features (X), labels (y), and project groups
8.	FOR each project split
9.	Train data = all project expect one
10.	Test data = one project
11.	Initialize MinMaxScaler
12.	Fit_transform on training data
13.	Transform on testing data
14.	END FOR
15.	**END**

D. Deep Neural Network

Deep Neural Network (DNN) is a deep learning approach characterized by a multilayered architecture, in which each layer consists of interconnected processing units. The DNN structure comprises three main components: an input layer, an output layer, and one or more hidden layers. The input layer receives the feature space derived from feature extraction, while the output layer is configured according to the classification type: a single node for binary classification or multiple nodes corresponding to the number of classes in multi-class classification [34].

In this study, the DNN model used as the baseline consists of three hidden layers with 64, 32, and 16 neurons, respectively, employing the ReLU activation function. This configuration is selected because deep learning operates hierarchically by decomposing complex functions into progressively simpler representations across each layer [35], while excessively complex architectures may lead to overfitting and reduce the model's generalization capability on datasets with limited feature dimensionality [36]. To validate the selection of this architecture, comparative experiments using deeper architectures are presented in Table 4.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Table 4. Comparison of DNN Architectures

Number of Hidden Layers	Number of Neurons in Each Layer	Avg. ROC-AUC
3	64 – 32 – 16	0.705
4	64 – 32 – 16 – 8	0.699
5	64 – 32 –16 – 8 – 4	0.674

Based on Table 4, the DNN architecture with three hidden layers achieves the best performance, whereas deeper architectures exhibit performance degradation due to model complexity that is disproportionate to the data dimensionality, which consists of only six input features. Each hidden layer is equipped with a dropout rate of 0.2 to prevent overfitting during training. The output layer consists of a single neuron with a sigmoid activation function to produce binary classification predictions. The training process is conducted using the Adam optimizer with a learning rate of 0.001, a batch size of 16, and 100 epochs. The predefined baseline hyperparameter configuration subsequently serves as the reference for the optimization process. The hyperparameter search space explored by the metaheuristic algorithms for the DNN model is presented in Table 5.

E. Particle Swarm Optimization

Particle Swarm Optimization (PSO) has been widely used to address optimization problems due to its simplicity and fast convergence [37]. PSO is inspired by the natural behavior of bird flocks when searching for food sources. Within the search space, each particle is characterized by its position and velocity, which are continuously updated at each iteration. These updates are governed by a predefined inertia weight, where each particle simultaneously considers two key pieces of information to guide its movement, namely the best solution it has individually achieved (personal best) and the best solution found by the entire swarm (global best). Through this iterative updating process, both individual particles and the swarm as a whole can explore the solution space more effectively and converge toward increasingly optimal solutions [38]. During the search process, the position and velocity of each particle are updated at each iteration using Eq. (2) and Eq. (3) [39].

$$v^{t+1}{}_i = w * v_i^t + c1r1\,(xBest_i^t - x_i^t) + c2r2\,(gBest_i^t - x_i^t) \quad (2)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3)$$

Based on Eq. (2), v_i^{t+1} represents the updated velocity of each particle, while v_i^t denotes the particle's velocity in the previous iteration. The parameter w is the inertia weight, c1 and c2 are the cognitive and social coefficients, and r1 and r2 are random values. The term $xBest_i^t$ refers to the best position previously found by the particle itself (personal best), whereas $gBest_i^t$ denotes the best position identified by the entire swarm (global best). The particle position at iteration t is represented by x_i^t. After obtaining the updated velocity, the particle position is subsequently updated using Eq. (3) [39]. In this study, PSO is configured with a population size of 20, a maximum of 10 iterations, and an inertia weight (w) of 0.4, a cognitive coefficient (c1) of 0.205, and a social coefficient (c2) of 2.205.

Table 5. Hyperparameter Search Space of Metaheuristic Optimization

Hyperparameter	Range
Neurons Layer 1	[8 – 128]
Neurons Layer 2	[8 – 128]
Neurons Layer 3	[8 – 128]
Dropout Rate	[0.1 – 0.5]
Learning Rate	[0.0001 – 0.01]
Batch Size	[16 – 128]

F. Genetic Algorithm

Inspired by the principles of natural selection and biological evolution, the Genetic Algorithm (GA) uses genetic mechanisms to explore the search space to identify optimal solutions. The search process begins by generating a population of chromosomes at random, where each chromosome represents a candidate solution evaluated by a fitness function. Subsequent generations are then produced through genetic operators, including selection, crossover, and mutation. This process is performed iteratively until a termination criterion is satisfied [40]. The steps of the Genetic Algorithm are as follows [41].

1. The search process in the Genetic Algorithm begins by generating an initial population of N chromosomes randomly, as described in Eq. (4) [41].

$$pop_i(t), t = 1, i = 1, 2, \dots, N \quad (4)$$

2. Each chromosome $tpop\ (t)$ in the population $pop\ (t)$ is then evaluated using a fitness function to assess its quality, as shown in Eq. (5) [41].

$$f_i = fitness(pop_i(t)) \quad (5)$$

3. The search process is evaluated based on convergence criteria. If the criteria are not satisfied, the algorithm proceeds to the next stage.
4. The selection operator selects the best chromosomes based on each individual's selection probability, as described in Eq. (6) [41].

$$P_i = \frac{f_i}{\sum_{i=1}^{N} f_i}, i = 1, 2, 3, \dots, N \quad (6)$$

Individuals are then randomly selected from this probability distribution to form a new population, as expressed in Eq. (7) [41].

$$newpop(t+1) = \{pop_j(t) | (j = 1, 2, \dots, N)\} \quad (7)$$

5. After the selection process, crossover is performed to generate a new set of chromosomes $tcrosspop(t+1)$. This process involves pairing selected individuals based on the crossover probability P_c.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

6. The final stage is mutation, in which genes on chromosomes are randomly altered with a small probability P_m. The resulting population forms $tmutpop\ (t+1)$, which is then defined as the new population: $pop\ (t) = mutpop\ (t+1)$. The process subsequently returns to Step 2 and iterates until the convergence criteria are satisfied.

The configuration of the Genetic Algorithm used in this study includes a population size of 20, a maximum of 10 iterations, a crossover probability of 0.95, and a mutation probability of 0.025.

G. Grey Wolf Optimization

Analogous to gray wolves that hunt in a coordinated manner with a defined leadership hierarchy, the Grey Wolf Optimizer (GWO) adopts this hierarchical structure and hunting strategy to search for optimal solutions in various optimization problems [42]. The algorithm categorizes the population into four hierarchical levels, namely alpha, beta, delta, and omega. The alpha wolf represents the best solution identified, while the beta and delta wolves correspond to the second- and third-best solutions, respectively, and the omega wolves occupy the lowest rank, representing the remaining population [43]. The hunting strategy of gray wolves begins with the encircling phase of the prey, which is mathematically modeled through Eq. (8) and Eq. (9) [39].

$$\vec{D} = \left|\vec{C}.\vec{X}_p(t) - \vec{X}(t)\right| \quad (8)$$

$$\vec{X}(t+1) = \vec{X}_p(t) = \vec{A}.\vec{D} \quad (9)$$

where $\vec{X}_p$ represents the position vector of the prey, $\vec{X}$ denotes the position vector of the gray wolf, and t indicates the current iteration. The coefficient vectors $\vec{A}$ and $\vec{C}$ are computed using Eq. (10) and Eq. (11) [39].

$$\vec{A} = 2\vec{a}.\vec{r}_1 - \vec{a} \quad (10)$$

$$\vec{C} = 2\vec{r_2} \quad (11)$$

where r_1 and r_2 are random values within the range of 0 to 1, and $\vec{a}$ is a parameter that decreases linearly from 2 to 0 over the course of iterations. To simulate the hunting behavior, the alpha (α), beta (β), and delta (δ) Wolves are assumed to have the best knowledge of the prey's position. The positions of these three wolves are updated using Eqs. (12), (13), and (14) [39].

$$\begin{aligned}\overrightarrow{D_\alpha} &= \left|\overrightarrow{C_1}.\overrightarrow{X_\alpha} - \vec{X}\right|,\\ \overrightarrow{D_\beta} &= \left|\overrightarrow{C_2}.\overrightarrow{X_\beta} - \vec{X}\right|,\\ \overrightarrow{D_\delta} &= \left|\overrightarrow{C_3}.\overrightarrow{X_\delta} - \vec{X}\right|\end{aligned} \quad (12)$$

$$\begin{aligned}\overrightarrow{X_1} &= \overrightarrow{X_\alpha} - \overrightarrow{A_1}.\left(\overrightarrow{D_\alpha}\right),\\ \overrightarrow{X_2} &= \overrightarrow{X_\beta} - \overrightarrow{A_2}.\left(\overrightarrow{D_\beta}\right),\\ \overrightarrow{X_3} &= \overrightarrow{X_\delta} - \overrightarrow{A_3}.\left(\overrightarrow{D_\delta}\right)\end{aligned} \quad (13)$$

$$\vec{X}_{(t+1)} = \frac{\overrightarrow{X_1} + \overrightarrow{X_2} + \overrightarrow{X_3}}{3} \quad (14)$$

Gray wolves attack the prey once it stops moving to complete the hunting process. The condition $\left|\vec{A}\right| < 1$ forces the wolves to attack the prey, while $\vec{C}$ helps prevent the solution from becoming trapped in local optima by encouraging exploration of the search space. This process continues until the termination criterion is satisfied [39]. In this study, the GWO experiments are conducted with a population size of 20 and a maximum of 10 iterations.

H. Performance Evaluation

In this study, the Receiver Operating Characteristic Area Under the Curve (ROC-AUC) is used as the primary metric to evaluate overall model performance. ROC-AUC is selected for its superior ability to handle imbalanced class distributions, making it a highly reliable and representative metric for assessing classification performance [44]. ROC-AUC ranges from 0 to 1, where each value reflects the model's ability to distinguish between classes. An ROC-AUC value of 1 indicates perfect classification performance, whereas an ROC-AUC value of 0.5 indicates that the model performs no better than random classification [45]. ROC-AUC metric is formulated in Eq. (15) [45].

$$AUC = \int_0^1 TPR\left(FPR^{-1}(t)\right)dt \quad (15)$$

In this equation, the AUC is defined as the area under the curve formed by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR), while t represents the different classification threshold values used in the evaluation process [46]. In addition to ROC-AUC as the primary evaluation metric, several supplementary metrics, including F1-score, Matthews Correlation Coefficient (MCC), and PR-AUC, are also employed to provide a more comprehensive perspective on model performance. These metrics are utilized to complement the analysis, particularly in evaluating the balance between precision and recall, as well as the model's performance under class imbalance conditions. Nevertheless, the primary focus of evaluation in this study remains on ROC-AUC.

I. Wilcoxon Signed Rank Test

In this study, the Wilcoxon Signed-Rank Test is employed to examine whether there are statistically significant differences between the DNN model with metaheuristic-based hyperparameter tuning and the model without hyperparameter tuning. The Wilcoxon Signed-Rank Test is a statistical method that can be applied to paired data without requiring the data to follow a specific distribution, making it suitable for relatively small datasets [47]. At the 95% confidence level, a difference between methods is considered statistically significant if the p-value is less than 0.05. Conversely, if the p-value is 0.05 or greater, the difference between the methods is considered not statistically significant [48].

III. Results

This study produces three experimental combinations based on PSO, GA, and GWO, which are employed to

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

optimize DNN hyperparameters within a CPDP framework across 14 open-source Android projects. To ensure the reproducibility and consistency of the experimental results, all experiments are conducted under a deterministic configuration using TensorFlow, where the random seed is globally initialized through tf.keras.utils.set_random_seed(42), which automatically sets the seed for Python, NumPy, and TensorFlow simultaneously. In addition, deterministic operations are enabled through tf.config.experimental.enable_op_determinism() to ensure that all operations produce consistent outputs. Prior to hyperparameter tuning, the DNN is first evaluated as a baseline for comparison. The best DNN hyperparameter configurations identified by each metaheuristic algorithm are presented in Table 6, while the ROC-AUC results of all combinations, along with the baseline model, are reported in Table 7.

Table 6. Best Hyperparameter Configuration Found by Metaheuristic Algorithm

Hyperparameter	PSO	GA	GWO
Neurons Layer 1	62	115	107
Neurons Layer 2	114	126	112
Neurons Layer 3	22	107	125
Dropout Rate	0.1	0.1	0.1
Learning Rate	0.002	0.001	0.002
Batch Size	64	64	16

Based on Table 7, the experimental results indicate that hyperparameter tuning with metaheuristic algorithms improves DNN performance compared to the baseline model. The best-performing combination is GWO-DNN, achieving an average ROC-AUC of 0.721, followed by GA-DNN with an average ROC-AUC of 0.715 and PSO-DNN with an average ROC-AUC of 0.714. The highest ROC-AUC improvement over the baseline is achieved by the GWO-DNN combination, with an increase of 0.016. The distribution of ROC-AUC values across the 14 projects is further visualized using a boxplot in Fig. 2, which shows that GWO-DNN exhibits the highest median ROC-AUC with a relatively more stable distribution compared to the other methods, indicating more consistent performance across projects.

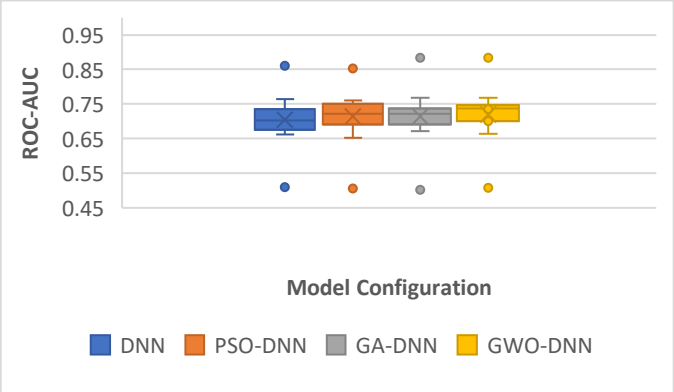

Fig. 2. Boxplot of ROC-AUC Scores

In addition to ROC-AUC, the performance of each combination is further evaluated using F1-Score, MCC, and PR-AUC as supporting metrics. The average values of these metrics for all combinations are summarized in Table 8. Based on Table 8, the performance improvements across the supporting metrics exhibit variation among the methods. In terms of F1-Score, GA-DNN demonstrates the most notable improvement at 0.347, followed by GWO-DNN at 0.340, while PSO-DNN does not show a meaningful change compared to the baseline at 0.314. These findings indicate that hyperparameter optimization exerts different effects on the balance between precision and recall. Regarding MCC, GA-DNN provides the best improvement at 0.268, followed by GWO-DNN at 0.254 and the baseline DNN at 0.252, while PSO-DNN experiences performance degradation with an MCC of 0.242. The less consistent MCC pattern compared to other metrics indicates that performance improvements are not always uniformly

Table 7. ROC-AUC Performance of All Combinations on Each Project

Dataset	Method			
	DNN	PSO-DNN	GA-DNN	GWO-DNN
Afwall	0.693	0.694	0.696	0.708
Alfresco	0.698	0.744	0.718	0.742
androidSync	0.725	0.736	0.730	0.739
androidWallpaper	0.667	0.652	0.672	0.664
anySoftKeyboard	0.729	0.717	0.719	0.737
Apg	0.678	0.719	0.725	0.709
atmosphere	0.662	0.684	0.678	0.701
chatSecure	0.727	0.750	0.733	0.748
facebook	0.751	0.756	0.751	0.746
flutter	0.510	0.506	0.503	0.508
kiwis	0.706	0.708	0.704	0.700
owncloudandroid	0.701	0.723	0.728	0.736
Pageturner	0.764	0.761	0.769	0.769
reddit	0.861	0.852	0.883	0.883
Average	0.705	0.714	0.715	0.721

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

reflected across all evaluation metrics. In terms of PR-AUC, GA-DNN and GWO-DNN achieve the highest values at 0.489, followed by PSO-DNN at 0.484 and the baseline DNN at 0.480. Given that PR-AUC is more sensitive to class imbalance, these results indicate that hyperparameter optimization can enhance the model's capability to detect minority classes.

Table 8. Performance Evaluation Results Using F1-Score, MCC, and PR-AUC

Method	F1-Score	MCC	PR-AUC
DNN	0.314	0.252	0.480
PSO-DNN	0.314	0.242	0.484
GA-DNN	0.347	0.268	0.489
GWO-DNN	0.340	0.254	0.489

IV. Discussion

The experimental results indicate that hyperparameter tuning using metaheuristic algorithms improves the performance of the DNN compared to the baseline model within a CPDP framework. The best performance is achieved by GWO-DNN, with an average ROC-AUC of 0.721, representing an improvement of 0.016 over the baseline model, which attains an average ROC-AUC of 0.705. Although the observed ROC-AUC improvement is moderate, it remains practically meaningful in the context of Software Defect Prediction (SDP). In CPDP scenarios characterized by significant class imbalance, each increase in ROC-AUC reflects an enhanced capability of The model to distinguish between defective and clean commits may ultimately contribute to improved efficiency in software testing processes. The performance differences among methods indicate that the selection of a metaheuristic algorithm influences the quality of the resulting hyperparameters. GWO-DNN demonstrates the best performance, suggesting more effective exploration and exploitation of the search space compared to PSO and GA. In contrast, PSO-DNN and GA-DNN do not exhibit statistically significant improvements over the baseline, indicating that the effectiveness of metaheuristic algorithms depends on the characteristics of the problem and the associated search space.

Table 9. Wilcoxon Signed-Rank Test and Effect Size Results

Method Comparison	p-Value	Significant (0.0083)	Cliff's Delta	Effect Size
PSO-DNN vs DNN	0.1530	No	0.1429	Negligible
GA-DNN vs DNN	0.0418	No	0.1633	Small
GWO-DNN vs DNN	0.0067	Yes	0.2653	Small
PSO-DNN vs GA-DNN	1.0000	No	0.0408	Negligible
PSO-DNN vs GWO-DNN	0.0905	No	-0.051	Negligible
GA-DNN vs GWO-DNN	0.1188	No	-0.1327	Negligible

To determine whether the observed performance improvements are statistically significant, the Wilcoxon Signed-Rank Test is applied to the ROC-AUC values obtained from the 14 projects with a significance level of ($p < 0.05$). Considering the presence of multiple comparisons in this analysis, the Bonferroni correction is employed, resulting in an adjusted significance threshold of ($p < 0.0083$). In addition, the effect size measured using Cliff's delta is included to assess the practical significance of the observed performance differences. The results of the statistical analysis are presented in Table 9.

Based on Table 9, after applying the Bonferroni correction, only the GWO-DNN combination demonstrates a statistically significant difference compared to the baseline model, with a p-value of 0.0067. Meanwhile, the GA-DNN and PSO-DNN combinations do not exhibit statistically significant differences. The Cliff's delta value for GWO-DNN is 0.2653, indicating a small effect size. This suggests that although the observed difference is statistically significant, its practical impact remains relatively limited. In addition to predictive performance, the computational cost of each metaheuristic algorithm is also analyzed to provide a more comprehensive perspective on the efficiency of each approach. Table 10 presents a comparison of the computational time required by each model combination during the hyperparameter optimization process. Based on Table 10, GWO-DNN achieves the best predictive performance, although it incurs the highest computational cost. In contrast, GA-DNN exhibits the lowest optimization and total computational time, followed by PSO-DNN, indicating the presence of a trade-off between predictive accuracy and computational efficiency. The differences in training time across models are attributable to variations in the hyperparameters each algorithm produces.

Table 10. Computational Cost Comparison

Method	Training Time (min)	Optimization Time (min)	Total Time (min)
PSO-DNN	25	10.631	10.656
GA-DNN	22	6.479	6.501
GWO-DNN	95	14.763	14.858

Further analysis of the project-level results reveals that the flutter project consistently exhibits substantially lower performance than the other projects across all evaluated combinations, with ROC-AUC values ranging from 0.503 to 0.510, which are close to random classification. This outcome can be directly attributed to the characteristics of

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

the flutter dataset, which has an extremely low defect ratio of 1.2%. As a result, the model encounters difficulty in effectively learning the patterns of defective commits due to the limited availability of minority class samples. This condition indicates that severe class imbalance within a project can significantly limit the effectiveness of metaheuristic-based hyperparameter tuning, regardless of the algorithm employed. In contrast, projects with more balanced defect rates, such as Reddit and Pageturner, demonstrate better performance, reinforcing the argument that class distribution characteristics within the dataset constitute a major determining factor in the effectiveness of cross-project software defect prediction models.

The findings of this study are consistent with several previous studies that demonstrate the effectiveness of metaheuristic algorithms in DNN hyperparameter tuning. Mumtahina et al. [22] state that PSO, GA, and GWO are among the most widely adopted metaheuristic algorithms for hyperparameter tuning due to their effectiveness in exploring complex search spaces. Meanwhile, Saputri et al. [23] emphasize that these three algorithms possess strong capabilities in balancing exploration and exploitation within the search space, enabling them to produce more optimal hyperparameter configurations compared to conventional methods such as Grid Search.

A comparison between the results of this study and previous studies employing the ROC-AUC metric in the context of software defect prediction is presented in Table 12.

Based on Table 12, all combinations in this study outperform the comparative models, with GWO-DNN achieving the best performance at 0.721. In terms of similarity, all studies confirm that mobile application defect prediction remains challenging, with ROC-AUC values generally ranging from 0.5195 to 0.721, consistent with the observation by Jorayeva et al. [25] that there is still room for improvement in CPDP scenarios. Regarding differences, Du et al. [24] reported the lowest ROC-AUC of 0.5195, attributable to their focus on the more complex task of line-level defect localization rather than commit-level prediction. Zhao et al. [49] reported 0.636 using an ensemble-based cascade forest model, yet this remains lower than our results, indicating that metaheuristic-based hyperparameter optimization may offer a more effective performance improvement strategy than architectural modifications alone. Catolino et al. [27] achieved an ROC-AUC of 0.690 using SVM on the same 14-project dataset, and the higher values achieved in this study suggest that metaheuristic-based hyperparameter optimization of DNN contributes additional performance gains beyond those already provided by the underlying feature set shared between both studies.

To evaluate the statistical significance of the performance differences, the Wilcoxon Signed-Rank Test is applied exclusively against the models of Jorayeva et al. [25], as it requires paired observations from identical projects. Although Zhao et al. [49] and Du et al. [24] provide project-level results, their datasets consist of different project sets, making paired statistical comparison infeasible. Catolino et al. [27] employ the same 14-project dataset but report only the average ROC-AUC value without a project-level breakdown, which similarly precludes a paired statistical test. Jorayeva et al. is therefore the only study for which a valid paired comparison is feasible, as the project-level ROC-AUC values were obtained through replicated experimental settings. Considering the presence of nine comparisons, Bonferroni correction is applied, resulting in an adjusted significance threshold of ($p < 0.0056$). In addition, Cliff's delta is calculated to assess the practical significance of the observed performance differences. The results of the statistical analysis are presented in Table 11.

Table 11. Wilcoxon Signed-Rank Test and Effect Size Results Against Jorayeva et al.

Method Comparison	p-Value	Significant (0.0056)	Cliff's Delta	Effect Size
PSO-DNN vs ANN	0.0494	No	0.3571	Medium
PSO-DNN vs CNN	0.0001	Yes	0.7857	Large
PSO-DNN vs LSTM	0.1726	No	0.3163	Small
GA-DNN vs ANN	0.0203	No	0.3571	Medium
GA-DNN vs CNN	0.0001	Yes	0.8061	Large
GA-DNN vs LSTM	0.0203	No	0.3265	Small
GWO-DNN vs ANN	0,0107	No	0.4082	Medium
GWO-DNN vs CNN	0.0001	Yes	0.8163	Large
GWO-DNN vs LSTM	0.0107	No	0.3469	Medium

After applying the Bonferroni correction, only the comparisons against the CNN model remain statistically significant, whereas the comparisons with the ANN and LSTM models no longer meet the adjusted significance threshold. These findings indicate that although the proposed methods generally achieve higher ROC-AUC values, the observed performance improvements are moderate and not consistently significant across all comparative models. Nevertheless, 2Cliff's delta results indicate that most comparisons against ANN produce medium effect sizes, whereas comparisons against CNN demonstrate large effect sizes. In contrast, comparisons against LSTM generally fall within the small to medium

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

effect size categories. These findings suggest that the proposed approach still has meaningful practical impact, particularly compared with the CNN model, even though not all differences remain statistically significant after the Bonferroni correction.

Table 12. Comparison of ROC-AUC Results with Previous Studies

Study	Method	Avg. ROC-AUC
Du et al. [24]	CodeLineDL	0.5195
Zhao et al. [49]	SDF	0.636
Jorayeva et al. [25]	ANN	0.679
	CNN	0.571
	LSTM	0.684
Catolino et al. [27]	SVM	0.690
Our Research	PSO - DNN	0.714
	GA - DNN	0.715
	GWO - DNN	0.721

Although this study successfully demonstrates the effectiveness of metaheuristic-based hyperparameter tuning on DNN within a CPDP framework, several limitations should be acknowledged. The dataset used in this study exhibits substantial class imbalance, where the number of defective commits is considerably lower than that of clean commits across most projects. This condition may affect the model's ability to effectively capture patterns associated with defective commits, particularly in projects with extremely low defect rates, such as Flutter. Although the ROC-AUC metric has been employed to mitigate the impact of class imbalance during the evaluation process, explicit handling of class imbalance during the preprocessing stage remains an important aspect to be considered in future research.

The findings of this study provide several important implications for practitioners and researchers in the field of software engineering. First, the results indicate that metaheuristic-based hyperparameter tuning is a sufficiently effective and practical approach for improving DNN performance in software defect prediction, particularly under conditions of limited historical data, which are common in software development organizations. However, the degree of improvement remains moderate and may vary depending on the characteristics of the dataset used. Second, the comparative analysis demonstrates that GWO is the most optimal metaheuristic algorithm for DNN hyperparameter tuning within a CPDP framework, thereby providing preliminary guidance for practitioners on selecting the most suitable algorithm. Third, the application of the Wilcoxon Signed-Rank Test provides strong statistical evidence that the observed performance improvements are not attributable to chance, thereby strengthening confidence in the proposed approach for practical application in real-world software development environments.

V. Conclusion

This study aims to analyze and compare the effectiveness of metaheuristic algorithm combinations, namely PSO, GA, and GWO, in optimizing DNN hyperparameters within a CPDP framework through a hyperparameter tuning process evaluated using the ROC-AUC metric across 14 open-source Android projects. The results demonstrate that metaheuristic-based hyperparameter tuning can improve performance compared to the baseline model, although the observed improvements are generally moderate. The best-performing combination is GWO-DNN, achieving an average ROC-AUC of 0.721, which represents an improvement of 0.016 over the baseline. The results of the Wilcoxon Signed-Rank Test indicate that only GWO-DNN achieves a statistically significant performance improvement after the application of the Bonferroni correction, whereas the remaining combinations do not exhibit significant differences. These findings suggest that the effectiveness of the proposed approach is contextual and may depend on the characteristics of the dataset employed. In addition, the analysis indicates that model performance is significantly influenced by the degree of data imbalance, where datasets with extreme class distributions, such as the Flutter project, exhibit low performance across all methods. These findings confirm that hyperparameter tuning alone is insufficient to address the challenges posed by highly imbalanced datasets. For future research, it is recommended to apply SMOTE (Synthetic Minority Over-sampling Technique) in the experimental process to address the significant class imbalance across projects. This condition may affect the model's ability to consistently capture patterns of defective commits. By applying SMOTE to synthetically increase the minority-class samples, the model is expected to better identify defective commits, thereby improving cross-project software defect prediction performance. To support the reproducibility of this study, all implementation code, parameter configurations, and experimental logs are publicly available in the following GitHub repository: https://github.com/maulanaabdulrahman/Metaheuristic-Based-Hyperparameter-Optimization-Analysis-of-DNN-for-CPDP-in-Mobile-Applications.git.

Acknowledgement

The authors would like to express their sincere gratitude to the Department of Computer Science, Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University, Banjarbaru, for the support and facilities provided throughout this research. The academic environment, available resources, and encouragement from members of the study program greatly contributed to this study. The authors also appreciate the institution's commitment to fostering research and innovation, which played an important role in the development of this work.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Data Availability
The datasets used in this study consist of publicly available Android mobile application open-source projects and can be accessed at: https://figshare.com/s/9a075be3e1fb64f76b48?file=14256152.

Author Contribution
Maulana Abdul Rahman contributed to the preparation of the manuscript, conducted the literature review, implemented the experimental framework, and analyzed and interpreted the research findings. Rudy Herteno was responsible for dataset preparation and curation, provided methodological insights, and supervised the overall execution of the study. Radityo Adi Nugroho contributed to the design and validation of the classification models and offered technical suggestions to refine the experimental methodology. Friska Abadi and Setyo Wahyu Saputro participated in the critical review of the manuscript, assessed the research methodology, and provided constructive feedback to enhance the paper's clarity, organization, and scientific quality. All authors reviewed and approved the final manuscript and accepted responsibility for the integrity and accuracy of the research.

Declarations
Ethical Approval
This study did not involve human participants or animal subjects. Therefore, ethical approval was not required.

Consent for Publication Participants
Not applicable.

Competing Interests
The authors declare no competing interests.

References
[1] X. Cai, S. Geng, D. Wu, and J. Chen, "Unified integration of many-objective optimization algorithm based on temporary offspring for software defects prediction," *Swarm Evol. Comput.*, vol. 63, Jun. 2021, doi: 10.1016/j.swevo.2021.100871.

[2] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Syst. Appl.*, vol. 171, Jun. 2021, doi: 10.1016/j.eswa.2021.114637.

[3] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, Oct. 2021, doi: 10.1016/j.jss.2021.111026.

[4] D. S. Balasubramaniam and D. S. G. Gollagi, "Software defect prediction via optimal trained convolutional neural network," *Advances in Engineering Software*, vol. 169, Jul. 2022, doi: 10.1016/j.advengsoft.2022.103138.

[5] P. Y. P. Chan, J. Keung, and Z. Yang, "Identifying inconsistent software defect predictions with symmetry metamorphic relation pattern," *Journal of Systems and Software*, vol. 227, Sep. 2025, doi: 10.1016/j.jss.2025.112449.

[6] F. Jiang, X. Yu, Q. Hu, J. Liu, and J. Du, "An ensemble method using neighborhood granular combination entropy for software defect prediction," *Inf. Process. Manag.*, vol. 63, no. 2, Mar. 2026, doi: 10.1016/j.ipm.2025.104483.

[7] S. S. Nahid, Md. S. Islam, Md. A. Hossain, M. M. Alam, and M. Shoyaib, "OCMTL: Transfer learning by orthogonal core-extraction of a matrix and its application in cross-project defect prediction," *Alexandria Engineering Journal*, vol. 133, pp. 498–516, Dec. 2025, doi: 10.1016/j.aej.2025.11.039.

[8] M. Azzeh and M. J. Abdel-Rahman, "Cross-project software defects prediction using fuzzy embedding and deep learning," *Inf. Softw. Technol.*, vol. 190, Feb. 2026, doi: 10.1016/j.infsof.2025.107968.

[9] Y. Xing, X. Qian, Y. Guan, B. Yang, and Y. Zhang, "Cross-project defect prediction based on G-LSTM model," *Pattern Recognit. Lett.*, vol. 160, pp. 50–57, Aug. 2022, doi: 10.1016/j.patrec.2022.04.039.

[10] Z. M. Zain, S. Sakri, and N. H. A. Ismail, "Application of Deep Learning in Software Defect Prediction: Systematic Literature Review and Meta-analysis," Jun. 01, 2023, *Elsevier B.V.* doi: 10.1016/j.infsof.2023.107175.

[11] Q. Liang, "Research on Software Defect Prediction Model based on Deep Learning," *Highlights in Science, Engineering and Technology*, vol. 122, pp. 23–29, Dec. 2024, doi: 10.54097/y0w76b47.

[12] S. R. Goyal, "Effective software defect prediction with deep neural networks," *Results in Engineering*, vol. 29, Mar. 2026, doi: 10.1016/j.rineng.2025.108378.

[13] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Sep. 2020, [Online]. Available: http://arxiv.org/abs/2002.08155

[14] C. Pan, M. Lu, and B. Xu, "An empirical study on software defect prediction using codebert model," *Applied Sciences (Switzerland)*, vol. 11, no. 11, Jun. 2021, doi: 10.3390/app11114793.

[15] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient Transformers: A Survey," Mar. 2022, [Online]. Available: http://arxiv.org/abs/2009.06732

[16] R. Dharani and K. Danesh, "Optimized deep learning ensemble for accurate oral cancer detection using CNNs and metaheuristic tuning,"

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

Intell. Based. Med., vol. 11, Jan. 2025, doi: 10.1016/j.ibmed.2025.100258.

[17] M. O. Miah, U. Habiba, and M. F. Kabir, "ODL-BCI: Optimal deep learning model for brain-computer interface to classify students confusion via hyperparameter tuning," *Brain Disord.*, vol. 13, Mar. 2024, doi: 10.1016/j.dscb.2024.100121.

[18] P. I. Kafilla, F. S. Utomo, and G. Karyono, "Enhancing Customer Purchase Behavior Prediction Using PSO-Tuned Ensemble Machine Learning Models," *Jurnal Teknik Informatika (Jutif)*, vol. 6, pp. 3509–3525, Oct. 2025, doi: 10.52436/1.jutif.2025.6.5.4952.

[19] F. Zito, E. G. Talbi, C. Cavallaro, V. Cutello, and M. Pavone, "Metaheuristics in automated machine learning: Strategies for optimization," *Intelligent Systems with Applications*, vol. 26, Jun. 2025, doi: 10.1016/j.iswa.2025.200532.

[20] A. H. Alharbi, E. S. M. El-kenawy, F. H. Rizk, K. S. Gaber, D. S. Khafaga, and M. M. Eid, "Optimized machine learning for building energy prediction: Feature selection and hyperparameter tuning using the Al-Biruni Earth Radius (BER) search optimization algorithm," *Energy Reports*, vol. 14, pp. 5505–5538, Dec. 2025, doi: 10.1016/j.egyr.2025.11.104.

[21] T. P. Nguyen, "AIoT-based indoor air quality prediction for building using enhanced metaheuristic algorithm and hybrid deep learning," *Journal of Building Engineering*, vol. 105, Jul. 2025, doi: 10.1016/j.jobe.2025.112448.

[22] U. Mumtahina, S. Alahakoon, and P. Wolfs, "Hyperparameter Tuning of Load-Forecasting Models Using Metaheuristic Optimization Algorithms—A Systematic Review," Nov. 01, 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/math12213353.

[23] T. R. D. Saputri, E. Kurniawan, C. C. Lestari, and T. Antonio, "Nature-based Hyperparameter Tuning of a Multilayer Perceptron Algorithm in Task Classification: A Case Study on Fear of Failure in Entrepreneurship," *Journal of Applied Data Sciences*, vol. 6, no. 2, pp. 969–980, May 2025, doi: 10.47738/jads.v6i2.539.

[24] X. Du, X. Yuan, and Z. Qiao, "Just-in-time software defect location for mobile applications based on pre-trained programming encoder model and hierarchical attention network," *Eng. Appl. Artif. Intell.*, vol. 148, May 2025, doi: 10.1016/j.engappai.2025.110381.

[25] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, "Deep Learning-Based Defect Prediction for Mobile Applications," *Sensors*, vol. 22, no. 13, Jul. 2022, doi: 10.3390/s22134734.

[26] X. Ju, Y. Cao, X. Chen, L. Gong, V. Chakma, and X. Zhou, "JIT-CF: Integrating contrastive learning with feature fusion for enhanced just-in-time defect prediction," *Inf. Softw. Technol.*, vol. 182, Jun. 2025, doi: 10.1016/j.infsof.2025.107706.

[27] G. Catolino, D. Di Nucci, and F. Ferrucci, "Cross-project just-in-time bug prediction for mobile apps: An empirical assessment," in *Proceedings - 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2019*, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 99–110. doi: 10.1109/MOBILESoft.2019.00023.

[28] Y. Wang and L. Singh, "Analyzing the impact of missing values and selection bias on fairness," *Int. J. Data Sci. Anal.*, vol. 12, no. 2, pp. 101–119, Aug. 2021, doi: 10.1007/s41060-021-00259-z.

[29] E. D. Prasetyo, B. Rahmat, and A. P. Sari, "Classification Of Cyber Attack And Anomaly In Web Server Using Transformer and Transfer Learning," *Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 7, no. 4, pp. 713–723, Nov. 2025, doi: 10.35882/ijeeemi.v7i4.119.

[30] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A survey on missing data in machine learning," *J. Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00516-9.

[31] A. M. Sharifnia, D. E. Kpormegbey, D. K. Thapa, and M. Cleary, "A Primer of Data Cleaning in Quantitative Research: Handling Missing Values and Outliers," *J. Adv. Nurs.*, vol. 82, no. 1, pp. 970–975, Jan. 2026, doi: 10.1111/jan.16908.

[32] D. Kartini, R. A. Badali, M. Muliadi, D. T. Nugrahadi, F. Indriani, and S. W. Saputro, "Dimensionality Reduction Using Principal Component Analysis and Feature Selection Using Genetic Algorithm with Support Vector Machine for Microarray Data Classification," *Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 7, Mar. 2025, doi: 10.35882/mr7x9713.

[33] H. Henderi, "Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer," *IJIIS: International Journal of Informatics and Information Systems*, vol. 4, pp. 13–20, Mar. 2021, doi: 10.47738/ijiis.v4i1.73.

[34] G. Giray, K. E. Bennin, Ö. Köksal, Ö. Babur, and B. Tekinerdogan, "On the use of deep learning in software defect prediction," *Journal of Systems and Software*, vol. 195, Jan. 2023, doi: 10.1016/j.jss.2022.111537.

[35] Z. Allen-Zhu and Y. Li, "Backward Feature Correction: How Deep Learning Performs Deep

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

(Hierarchical) Learning," Jul. 2023, [Online]. Available: http://arxiv.org/abs/2001.04413

[36] H. Sun, H. Zhang, M. Song, S. Li, and Y. Lu, "Automatic Fresnel zone picking in the dip-angle domain using deep neural networks," *Journal of Geophysics and Engineering*, vol. 16, no. 1, pp. 136–145, Feb. 2019, doi: 10.1093/jge/gxy012.

[37] M.-W. Li, Y.-T. Wang, S.-J. Yan, J. Geng, and X.-H. Wang, "An optimization method for tidal current turbine array layout based on particle swarm optimization," *Energy*, vol. 348, p. 140583, Apr. 2026, doi: 10.1016/j.energy.2026.140583.

[38] S. Gupta and K. Gautam, "Enhanced cooperative learning and local search integrated particle swarm optimization for global optimization and coverage enhancement in wireless sensor network," *Swarm Evol. Comput.*, vol. 100, Jan. 2026, doi: 10.1016/j.swevo.2025.102262.

[39] M. A. K. Raiaan *et al.*, "A systematic review of hyperparameter optimization techniques in Convolutional Neural Networks," Jun. 01, 2024, *Elsevier Inc.* doi: 10.1016/j.dajour.2024.100470.

[40] A. Yilmaz and İ. Kuş, "General CNN model for biomedical image classification via genetic algorithm-based hyperparameter optimization," *Ain Shams Engineering Journal*, vol. 17, no. 1, Jan. 2026, doi: 10.1016/j.asej.2025.103891.

[41] H. Zhi and S. Liu, "Face recognition based on genetic algorithm," *J. Vis. Commun. Image Represent.*, vol. 58, pp. 495–502, Jan. 2019, doi: 10.1016/j.jvcir.2018.12.012.

[42] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Enhancing Software Defect Prediction through Hybrid Optimization for Feature Selection and Gradient Boosting Classification," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 169–181, Apr. 2024, doi: 10.35882/jeeemi.v6i2.388.

[43] M. Rastgou, Y. He, R. Lou, and Q. Jiang, "A comparison of metaheuristic optimizations with automated hyperparameter tuning methods in support vector machines algorithm for predicting soil water characteristic curve," *Eng. Geol.*, vol. 353, Jun. 2025, doi: 10.1016/j.enggeo.2025.108121.

[44] D. Veganzones, E. Séverin, and S. Ben Jabeur, "Forecasting corporate bankruptcy in imbalanced datasets using a new hybrid machine learning approach," *Res. Int. Bus. Finance*, vol. 81, Jan. 2026, doi: 10.1016/j.ribaf.2025.103200.

[45] P. Nabella, R. Herteno, S. W. Saputro, M. R. Faisal, and F. Abadi, "Impact of a Synthetic Data Vault for Imbalanced Class in Cross-Project Defect Prediction," *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 219–230, Apr. 2024, doi: 10.35882/jeeemi.v6i2.409.

[46] S. Dubey, G. Tiwari, S. Singh, S. Goldberg, and E. Pinsky, "Using machine learning for healthcare treatment planning," *Front. Artif. Intell.*, vol. 6, 2023, doi: 10.3389/frai.2023.1124182.

[47] D. R. Widiana, Syafiuddin, I. Sriwijayasih, I. R. Aju, Y. Praharsi, and E. Novianarenty, "Penerapan Uji Wilcoxon Signed Rank Test Untuk Menganalisis Perbedaan Nilai Test Sebelum Dan Setelah Pelatihan Digital Marketing," *Jurnal Teknologi Maritim*, vol. 8, no. 2, pp. 23–32, Oct. 2025, doi: 10.35991/jtm.v8i2.73.

[48] N. Zhang, S. Ying, W. Ding, K. Zhu, and D. Zhu, "WGNCS: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation," *Inf. Sci. (N. Y).*, vol. 570, pp. 545–576, Sep. 2021, doi: 10.1016/j.ins.2021.05.008.

[49] K. Zhao, Z. Xu, T. Zhang, and Y. Tang, "Simplified Deep Forest Model based Just-In-Time Defect Prediction for Android Mobile Apps," pp. 222–222, Dec. 2020, doi: 10.1109/QRS51102.2020.00039.

Author Biography

Maulana Abdul Rahman is an undergraduate student in the Computer Science program at Lambung Mangkurat University. His academic interests include software defect prediction, deep learning, and metaheuristic-based optimization. His current research focuses on analyzing the effectiveness of metaheuristic-based hyperparameter tuning for deep neural networks in cross-project software defect prediction scenarios, using open-source Android mobile application datasets. During his studies, he has been actively involved in academic research to strengthen his technical competencies and analytical capabilities in software engineering. He can be contacted via email at 2211016110012@mhs.ulm.ac.id

Rudy Herteno obtained his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011 and gained professional experience as a software developer, contributing to the development of various software applications, particularly to support the operational needs of local government institutions. He completed his master's degree in Informatics at STMIK Amikom University in 2017. He is currently a lecturer in the Computer Science program at the Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University. His research interests include software engineering and deep learning, with a focus on improving software quality and developing artificial intelligence-

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340

based solutions. He can be contacted via email at rudy.herteno@ulm.ac.id

Radityo Adi Nugroho obtained his bachelor's degree in Informatics from the Islamic University of Indonesia in 2004 and completed his Master of Computer Science at Gadjah Mada University in 2007. He currently holds the position of Associate Professor in the Computer Science program at Lambung Mangkurat University. In addition to his academic role, he has extensive leadership experience, having served as the Head of the Information and Communication Technology Center at Lambung Mangkurat University from 2014 to 2023. His research interests include software defect prediction and computer vision, with a focus on improving system reliability and optimizing visual data processing across various applications. He can be contacted via email at radityo.adi@ulm.ac.id

Friska Abadi obtained his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011 and completed his Master's degree in Informatics at STMIK Amikom Yogyakarta in 2016. He is currently a lecturer in the Computer Science program at the Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University. In addition to his academic role, he serves as the Head of the Software Engineering Laboratory. As a practitioner in web and mobile application development, he has research interests in software engineering, data mining, and machine learning. He is actively involved in research and community service activities to support technological innovation. He can be contacted via email at friska.abadi@ulm.ac.id

Setyo Wahyu Saputro obtained his bachelor's degree in Computer Science from Lambung Mangkurat University in 2011 and completed his Master's degree in Informatics at STMIK Amikom University in 2016. He is currently a lecturer in the Computer Science program at the Faculty of Mathematics and Natural Sciences, Lambung Mangkurat University, Banjarbaru. Since 2017, he has been actively engaged as an information technology practitioner and consultant, serving as a project manager and system analyst for various projects in both government and private sectors in South Kalimantan. His research interests include software engineering, human–computer interaction, and artificial intelligence applications. He can be contacted via email at setyo.saputro@ulm.ac.id

Corresponding author: Rudy Herteno, rudy.herteno@ulm.ac.id, Department of Computer Science, Faculty of Mathematics and Natural Science, Banjarbaru,Indonesia
DOI: https://doi.org/10.35882/ijeeemi.v8i2.340