FEATURE SELECTION USING THE ZEBRA OPTIMIZATION ALGORITHM FOR SOFTWARE FAULT PREDICTION: A STUDY ON THE BUGHUNTER DATASET

Ha Thi Minh Phuong[2], Dao Khanh Duy[2], Nguyen Do Anh Nhu[1], Hoang Thi Thanh Ha[1]*

[1]*The University of Danang – University of Economics, Vietnam*

[2]*The University of Danang - Vietnam-Korea University of Information and Communication Technology, Vietnam*

*Corresponding author: ha.htt@due.edu.vn

(Received: May 02, 2025; Revised: June 14, 2025; Accepted: June 21, 2025)

DOI: 10.31130/ud-jst.2025.23(9C).533E

Abstract - Software fault prediction focuses on identifying software modules that are most likely to contain faults before the testing stage, helping developers allocate quality assurance resources effectively and improve system reliability. A major challenge in SFP lies in redundant and irrelevant features within software fault datasets, which often lower the accuracy of predictive models. To address this, the study introduces a wrapper-based feature selection method using the Zebra Optimization Algorithm (ZOA). Experiments on nine BugHunter datasets show that the ZOA-based method consistently surpasses a baseline deep learning model trained on raw data, achieving higher F1-score, Precision, and Recall. The findings demonstrate that ZOA is effective in reducing feature redundancy and improving prediction performance. This research confirms the potential of ZOA in SFP, offering practical benefits for software development and opening new opportunities for further studies.

Key words - Software fault prediction; machine learning; Zebra Optimization Algorithm; BugHunter dataset

1. Introduction

Recently, software has become a foundational component across diverse domains, including commerce, education, and critical infrastructure in the contemporary digital era. As software systems expand in scale and complexity, the potential for faults correspondingly increases, posing substantial risks. Even minor faults can lead to severe consequences, particularly in high-stakes sectors such as finance, healthcare, and air traffic management. Accordingly, the early detection of software faults during the first stages of development is imperative to mitigate potential risks and to uphold the reliability and quality of software products.

Software fault prediction (SFP) was developed as an effective method that enables developers to identify potential fault components, thereby focusing testing and maintenance resources effectively. Nevertheless, a significant challenge lies in the inherent complexity of software fault datasets, which often contain redundant features and exhibit highly imbalanced class distributions. These factors adversely affect the performance and generalizability of predictive models. This issue is addressed through the implementation of feature selection (FS), which is regarded as a critical stage in reducing data dimensionality and enhancing model accuracy. There are three types of feature selection methods, including filter, wrapper, and embedded. While they have yielded some favourable outcomes, they continue to possess constraints, including their high computational complexity, limited generalization capacity, and susceptibility to local optima. Metaheuristic algorithms, including Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Grey Wolf Optimization (GWO), and Ant Colony Optimization (ACO) [1, 2] which have been extensively employed in FS as a result of their capacity to identify optimal solutions in intricate spaces. Therefore, this study suggests that ZOA [3], which is employed to improve the accuracy of SFP by conducting a global search and preventing local optima for feature selection.

Recently, Ferenc et al. [4] introduced the BugHunter dataset, which includes a substantially larger number of instances, thereby enabling machine learning and deep learning models to achieve enhanced performance. A diverse and realistic platform for testing the performance of FS methods and machine learning models is provided by BugHunter, which includes numerous software projects with corresponding features and their labels. One of the main challenges with the BugHunter dataset is dimensionality, which can negatively impact model performance and increase computational complexity. We apply Zebra Optimization Algorithm (ZOA) to select optimal features that yield high performance of SFP models. The primary goal of this paper is to assess the efficacy of ZOA in enhancing the precision of SFP models. The research findings will establish a scientific foundation for the creation of SFP tools that are effective, thereby assisting software organisations in the enhancement of product quality and the reduction of development costs. The performance of ZOA wrapper-based FS methods is compared against the baseline method which applies learning techniques to the original software fault datasets.

We conducted experiments using nine distinct BugHunter datasets. Additionally, we applied deep learning models, specifically Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Long Short-Term Memory (LSTM) to assess the performance of the SFP models and compare their predictive capabilities across different feature subsets. The experimental results demonstrate that the presented wrapper-based FS method using the ZOA consistently outperforms the baseline approach. Particularly, the average results for all the datasets, the accuracy increases by 0.5%, the recall increases by 0.7%, the F1-score increases by 2.1%, the AUC increases by 0.5%.

2. Related work

Recent research on SFP aims to enhance the ability to proactively identify defective software modules, thereby optimizing resource allocation and improving overall software quality. By leveraging historical data and machine learning techniques, SFP facilitates early fault detection during the software development lifecycle, which in turn reduces costs and increases system reliability [5]. For example, SFP models have been instrumental in prioritizing testing efforts by pinpointing high-risk code segments, thereby minimizing the time and effort re- quired for debugging and maintenance [6, 2]. Moreover, the integration of static code analysis with SFP has improved fault localization accuracy, allowing testing efforts to be more effectively focused on critical code areas [2]. These advancements highlight the pivotal role of SFP in modern software engineering, enabling developers to address faults more proactively and efficiently [7, 2].

Addressing class imbalance where defective modules are significantly outnumbered by non-defective ones remains a major challenge in SFP. To tackle this issue, recent studies have explored resampling techniques such as the Synthetic Minority Oversampling Technique (SMOTE), which improves model performance by balancing the class distribution [8, 5]. Feature selection has emerged as a crucial strategy for improving both model accuracy and computational efficiency by isolating the most relevant software metrics for fault prediction. Notably, hybrid metaheuristic approaches such as the integration of Gray Wolf Optimization with Harris Hawks Optimizer have achieved promising results in selecting optimal feature subsets while minimizing redundancy [9]. These nature-inspired algorithms efficiently traverse high-dimensional feature spaces to retain only highly predictive features, thereby enhancing the overall performance of SFP models [10, 11].

In addition, the advancement of bagging and filter-based feature selection FS methods has attracted considerable interest due to their ability to balance predictive accuracy with computational efficiency. In the context of SFP, bagging-based FS techniques have proven particularly effective, as they adapt dynamically to specific datasets and classifiers by evaluating feature subsets based on model performance [6]. On the other hand, filter-based methods such as those utilizing statistical measures like ANOVA or Pearson correlation are computationally efficient and less prone to overfitting, making them especially suitable for large-scale software projects [12]. Moreover, recent research has highlighted the importance of dynamic re-ranking strategies, which continuously prioritize features based on their evolving relevance during model training. These advancements underscore the essential role of FS in improving SFP model interpretability and mitigating the challenges posed by high-dimensional datasets.

The evaluation of SFP models that employ deep learning (DL) commonly focuses on performance metrics such as precision, sensitivity, accuracy, F1-score, and the area under the ROC curve (AUC). Recent empirical findings have shown that integrating DL models with optimal feature selection (FS) techniques yields notable improvements across these metrics compared to traditional methods. In particular, the application of the Grey Wolf Optimizer (GWO) to refine the feature selection process in DL-based SFP models has been reported to reduce false positive classification errors while enhancing the F1-score [13]. Moreover, the capacity to concentrate on critical code segments has been improved by the implementation of attention mechanisms in DL models, which has subsequently improved performance [14]. In order to guarantee reliable evaluations, researchers have implemented benchmark datasets, including the NASA Metric Data Program (MDP), to assess the performance of models in a variety of software initiatives [6, 14]. These studies underscore the significance of preprocessing steps, including data cleansing and SMOTE-based equalization, in order to reduce noise and class imbalance, which can distort performance metrics. Additionally, FS has enhanced the computational efficacy of DL models, resulting in a reduction in training time without sacrificing accuracy. These results underscore the synergy between DL, FS, and rigorous evaluation in fostering SFP, offering practical solutions to real-world software development challenges [15, 12].

3. Basic concepts of ZOA Algorithm

The Zebra Optimization Algorithm [3] is a swarm-based algorithm first introduced by Trojovská et al. It simulates how plains zebras find food and adopt defensive strategies against predators. The core concepts behind ZOA are exploration and exploitation, and the algorithm aims to balance these two factors to both discover new search spaces and converge into promising regions. To achieve this, ZOA operates in two stages:

1. **Food Searching Stage (Exploitation)**: Convergence to the Pioneer Zebra (PZ).

2. **Defensive Stage**: Includes local strategy (s1) and exploration strategy (s_2).

Stage 1: Exploitation by PZ

In this stage, each zebra (representing a solution) updates its position toward the Pioneer Zebra (i.e., the best current solution in the population):

$$X_i^{new} = X_i + r.(PZ - l.X_i)$$

where, $X_i \in R^m$: position of the *i-th* zebra; F(X_i): value of the corresponding objective function; *PZ*: best current solution in the population; *r*: random number in (0, 1); *l*: effect coefficient (typically 1 or 2).

Stage 2: Defense

In this stage, each zebra simulates a response when attacked by a predator. Based on probability, one of the two following strategies is chosen:

Strategy s1: Local Exploitation (Zigzag Escape) This strategy mimics how a zebra avoids predators by moving in a zigzag pattern:

$$X_i^{new} = X_i + R_\tau . (2r - 1). \left(1 - \frac{t}{T}\right). X_i$$

where, R: small constant (typically 0.01); r: random number in (0, 1); t: current iteration; T: total number of iterations.

This strategy performs small adjustments to the solution over time, helping fine-tune the position in later iterations.

Strategy s2: Exploration (Support Behaviour) This strategy simulates cooperative behaviour when a fellow zebra is under attack:

$$X_i^{new} = X_i + r(AZ - l. X_i) \quad (1)$$

where, AZ: a randomly selected individual from the population (attacked zebra); l: effect coefficient (as in Stage 1).

This strategy encourages movement into different regions of the search space, thus enhancing exploration. ZOA is known for its ease of implementation and low number of parameters. However, its simplicity also makes it susceptible to getting stuck in local optima and being sensitive to parameter choices. Therefore, several variants, such as Improved ZOA (IZOA) [16], have been proposed to address these limitations.

4. Experiment Design

In this research, we investigated the effectiveness of features selected by ZOA using three deep learning models: CNN, RNN, and LSTM. The details of the experimental design are presented in Figure 1. In the first stage, we collected nine projects from the BugHunter dataset. As mentioned in Section 4.1, the dataset is preprocessed by handling missing values and applying RobustScaler normalization. Subsequently, class imbalance is addressed using SMOTE and optimal features are selected by ZOA. Finally, the processed data were employed to train three deep learning models, whose performance was subsequently evaluated in terms of accuracy, AUC, F1-score and recall.

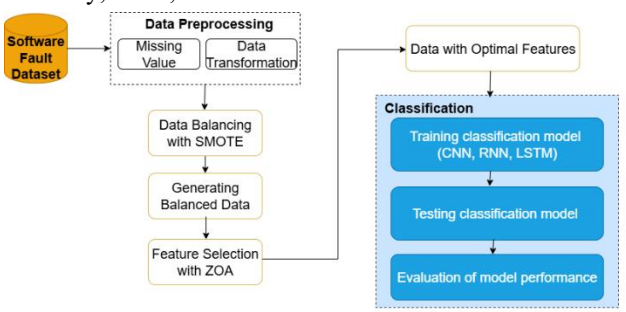

Figure 1. *The proposed methodology.*

4.1. Dataset

In addition to the widely used NASA dataset in software fault prediction research, the recently introduced BugHunter dataset offers a larger scale and provides new opportunities for exploring various research directions. Therefore, we collected nine projects from this source; the details of each project are presented in Table 1. Each project includes 98 features, such as Comment Rules, Coupling Rules, *etc* and one dependent attribute, "Number of Bugs". Given that the fault ratio in most projects is below 50%, this study employed an oversampling technique SMOTE to address the class imbalance issue.

Table 1. *The BugHunter datasets used in the study*

Dataset	Projects	Instance	Faulty Instance	Non-Faulty Instance	Faulty ratio (%)
BugHunter	ceylon-ide-eclipse	1477	520	957	35.21
	oryx	646	81	565	12.54
	titan	506	195	311	38.54
	orientdb	4770	2153	2617	45.14
	BroadleafCommerce	3241	1569	1672	48.41
	hazelcast	28185	14926	13259	52.96
	Junit				33.14
	Netty	338	112	226	39.72
	Elastic search	6591	2618	3973	49.83
		31644	15769	15875	

4.2. ZOA for optimization

To enhance model performance, we employ the ZOA to select the optimal subset of features. In the objective function, each candidate solution is encoded as a binary vector, where each bit indicates whether a feature is selected or discarded. To reduce computation time, Logistic Regression [17] is used in the objective function instead of more complex deep learning models such as CNN, RNN, or LSTM. The model is then validated on a hold-out validation set, and its performance is assessed using the F1-score. The optimization is carried out using the Mealpy library [18], with ZOA configured to run for 10 epochs and a population size of 40. However, the choice of 10 epochs and a population size of 40 was determined experimentally and may not be optimal. To address this issue, we aim to investigate and identify the optimal parameters for ZOA training to further enhance model performance.

4.3. Learners

CNN [19] is a well-known deep learning model that has been widely used in various applications, especially in computer vision, due to its ability to extract important features through convolutional networks. A typical CNN architecture includes convolutional layers, activation layers, pooling layers, fully connected layers, dropout layers, and batch normalization layers.

RNN [20] is a deep learning algorithm suitable for sequential data such as time-series or text, as it maintains hidden states to store information from previous steps. However, RNNs commonly face issues such as vanishing or exploding gradients, making it difficult to learn long sequences.

LSTM [21] is a variant of RNN designed to learn long sequences and address the vanishing gradient problem by incorporating forget, input, and output gates. It is commonly applied in tasks such as translation, time-series prediction, and natural language processing.

4.4. Performance Metrics

To thoroughly evaluate and understand the performance of the models, we focused on four main metrics: accuracy, F1-score, AUC, and recall.

– True Positive (TP): The number of positive samples classified correctly.

– True Negative (TN): The number of negative samples classified correctly.

– False Positive (FP): The number of negative samples classified as positive samples.

– False Negative (FN): The number of positive samples classified as negative samples.

Precision is the ratio of correctly predicted positive cases to the total number of cases predicted as positive. It is represented as:

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

Accuracy indicates how correct the model's predictions are. Its value ranges from 0 to 1, where an accuracy of 1 means the model is perfectly accurate.

Recall, also known as sensitivity, measures the proportion of actual positive samples that are correctly predicted by the model. It is represented as:

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

F1-score is calculated as:

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \quad (4)$$

The F1-score provides a balanced metric between Precision and Recall. It is especially useful for evaluating models on imbalanced datasets.

The AUC (Area Under the Curve) is used to assess the effectiveness of the SFP model. A curve that closely approaches the upper-left corner of the plot indicates strong model performance, while significant deviation from this region suggests weaker performance.

5. Experimental Results

Table 2 evaluates the functionality of three prevalent deep learning models: CNN, RNN, and LSTM. The BugHunter dataset is used to evaluate these models. Accuracy, Area Under the Curve (AUC), F1-score, and Recall are the performance metrics that are employed for comparison. The Oryx dataset yields the most outstanding performance across all models and evaluation metrics.

Notably, the LSTM model achieves values of accuracy, AUC, F1-score and recall of 84.6%, 93.9%, 85.7% and 91.6%, respectively. Additionally, the CNN and RNN models attain accuracy scores of 85.0% and 82.7%, respectively. In contrast, the Hazelcast datasets exhibit the lowest predictive performance among the evaluated datasets. The Hazelcast dataset had the lowest accuracy value with CNN at 52.1% and a subpar F1-score of 30.3%. The JUnit and BroadleafCommerce datasets demonstrated satisfactory performance, with accuracy values ranging from 68.0% to 73.2% and 66.5% to 66.7%. F1-scores as low as 50.7% were achieved with the LSTM model on Elasticsearch, while Titan and Elasticsearch were in the medium to low range.

Table 2. *Performance comparison of deep learning models*

Dataset	Model	Acc	AUC	F1-score	Recall	Feature
Ceylon-ide-eclipse	CNN	0.576	0.624	0.576	0.590	12
	RNN	**0.586**	0.619	**0.609**	**0.657**	
	LSTM	0.581	**0.621**	0.599	0.633	
OrientDB	CNN	0.600	0.646	0.569	0.538	49
	RNN	**0.603**	**0.650**	**0.612**	**0.629**	
	LSTM	0.597	0.646	0.558	0.511	
Titan	CNN	0.521	0.524	0.303	0.362	49
	RNN	**0.549**	**0.565**	**0.588**	**0.647**	
	LSTM	0.544	0.561	0.525	0.506	
Hazelcast	CNN	0.521	0.524	0.303	0.362	57
	RNN	**0.549**	**0.565**	**0.588**	**0.647**	
	LSTM	0.544	0.561	0.525	0.506	
JUnit	CNN	0.721	**0.784**	0.724	0.740	47
	RNN	**0.732**	0.767	0.709	0.701	
	LSTM	0.680	0.723	**0.700**	**0.746**	
Netty	CNN	0.571	0.604	**0.598**	**0.652**	31
	RNN	**0.565**	**0.600**	0.546	0.536	
	LSTM	0.557	0.586	0.518	0.480	
Broadleaf Commerce	CNN	0.667	0.726	0.635	0.582	32
	RNN	0.666	0.724	0.638	0.591	
	LSTM	0.665	**0.729**	0.632	0.577	
Oryx	CNN	0.850	0.921	0.853	0.871	58
	RNN	0.827	0.914	0.828	0.836	
	LSTM	**0.846**	**0.939**	**0.857**	**0.916**	
Elastic search	CNN	0.561	0.585	0.548	0.543	23
	RNN	**0.557**	**0.589**	**0.565**	**0.583**	
	LSTM	0.554	0.579	0.507	0.459	

The results indicate a substantial degree of performance variability among the datasets. Oryx obtains the most favourable outcomes because of its high-quality features and clean data, whereas Hazelcast fail to perform well due to suboptimal features or noise. Oryx dataset yields the most outstanding results across all models, with LSTM achieving the highest Accuracy value of 0.846, the AUC value of 0.939, the F1-score value of 0.857, and the Recall value of 0.916. The high number of informative features (58 features) and rich data structure appear to benefit from LSTM's capability to model long-term dependencies, highlighting the importance of context in predicting faults. For ElasticSearch dataset, the performance across models is generally low, with F1-scores around 0.507–0.583. Despite a moderate feature count (23 features), this result may be attributed to either poor feature relevance or a noisy dataset, limiting the models' ability to generalize. RNN exhibits high sensitivity to challenging data, LSTM performs well on complex sequential data, and CNN is appropriate for structured data and maintain performance after equalization. To enhance the results, the primary factor is to select the appropriate model and optimize features based on the characteristics of the data.

These results suggest that the consistently strong performance of RNN and CNN models highlights their

suitability for this domain. However, traditional machine learning approaches remain viable alternatives in specific scenarios, demonstrating competitive performance in certain metrics.

As shown in Figure 2, the presented feature selection method utilizing the ZOA consistently outperforms the baseline across multiple evaluation metrics. It achieves a 0.5% increase in accuracy, a 0.5% improvement in AUC, a notable gain of 2.1% in F1-score, and 0.7% in Recall. These results highlight the superior generalization capability of the ZOA-based method, especially in terms of F1-score and Recall, which are critical for imbalanced datasets commonly found in software fault prediction. The improvements indicate that ZOA enhances the model's ability to detect faulty modules while maintaining overall predictive performance.

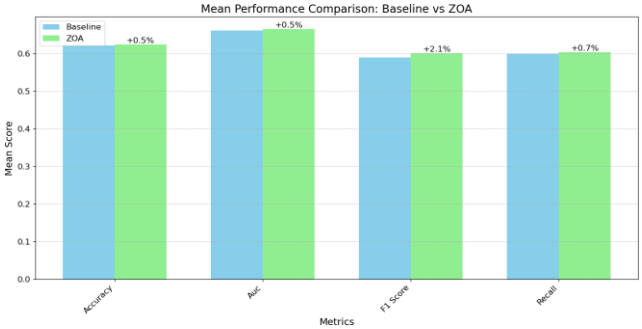

Figure 2. *The comparative results of the approach using ZOA for feature selection and the baseline method.*

To assess the effectiveness of the Zebra Optimization Algorithm (ZOA) in feature selection, we compared it against two other popular metaheuristic optimization algorithms: Particle Swarm Optimization (PSO) and Grey Wolf Optimizer (GWO). This comparison was conducted using the same BugHunter dataset. We evaluated the experimental results based on three crucial metrics: Accuracy, Area Under the ROC Curve (AUC), and F1-score.

Table 3. *Accuracy Comparison of Deep Learning Models with Different Optimization Algorithms*

Model	**Accuracy**		
	PSO	**GWO**	**ZOA**
CNN	**0.627**	0.615	0.625
Simple-RNN	0.625	0.624	**0.627**
LTSM	0.631	**0.632**	0.619

Table 4. *AUC Comparison of Deep Learning Models with Different Optimization Algorithms*

Model	**AUC**		
	PSO	**GWO**	**ZOA**
CNN	0.661	0.655	**0.665**
Simple-RNN	0.663	0.666	**0.667**
LTSM	0.669	**0.671**	0.662

Table 5. *F1-Score Comparison of Deep Learning Models with Different Optimization Algorithms*

Model	**F1-Score**		
	PSO	**GWO**	**ZOA**
CNN	**0.586**	0.560	0.575
Simple-RNN	0.622	0.616	**0.623**
LTSM	0.617	0.622	0.609

Table 6. *Computational time comparison different optimization algorithms for feature selection*

Optimization Algorithms	*Computational Time*
PSO	**118.9**
GWO	132.2
ZOA	265.8

Tables 3-5 present the performance of different deep learning models with PSO, GWO and ZOA in terms of accuracy, AUC and F1-score. The results clearly show that the algorithms' performance varies significantly depending on the machine learning model used. In Table 3, for the CNN model, PSO achieved the highest accuracy (0.627), outperforming ZOA (0.625) and GWO (0.615). In terms of AUC, ZOA led with 0.665, followed by PSO (0.661) and GWO (0.655). However, when considering the F1-score, PSO yielded the best result (0.586), significantly higher than GWO (0.560) and ZOA (0.575). With the SimpleRNN model, the ZOA algorithm demonstrated clear superiority with an accuracy of 0.627, surpassing PSO (0.625) and GWO (0.624). The AUC metric also showed ZOA in the lead with 0.667, followed by GWO (0.666) and PSO (0.663). For the F1-score, ZOA achieved the best result (0.623), outperforming GWO (0.616) and PSO (0.622). For the LSTM model, the results indicated intense competition among the algorithms. GWO achieved the highest accuracy (0.632), followed by PSO (0.631) and ZOA (0.619). However, regarding the critical AUC metric, GWO led with 0.671, followed by PSO (0.669) and ZOA (0.662). For the F1-score, GWO also produced the best result (0.622), higher than PSO (0.617) and ZOA (0.609).

Another important factor to consider is the computational efficiency of these algorithms. In Table 6, the average execution times revealed that PSO was the fastest at 118.90 seconds, followed by GWO at 132.21 seconds, and ZOA was the slowest at 265.81 seconds. ZOA's computation time was notably higher, taking 2.2 times longer than PSO and 2.0 times longer than GWO. This extended time is likely due to ZOA's more intricate search mechanism, which requires more calculations during the optimization process.

The experimental results show that no single algorithm completely dominates across all models and evaluation metrics. PSO demonstrated stable performance and was particularly excellent with the CNN model. GWO showed clear superiority on the LSTM model, achieving the highest evaluation scores. ZOA proved most suitable for the SimpleRNN model, delivering the best results across most metrics. This variation can be attributed to the unique characteristics of each optimization algorithm and how they interact with the different architectures of deep learning models.

A significant finding is the inverse relationship observed between performance and computation time. PSO not only had the fastest execution time but also achieved good performance across various metrics, making it an attractive choice for practical applications. GWO strikes a good balance between performance and computation time, especially for the LSTM model. ZOA,

despite its higher computation time, still shows potential in specific cases, such as with the SimpleRNN model. Therefore, selecting the appropriate algorithm requires careful consideration based on the specific application's requirements: prioritizing processing speed versus result quality, and the type of machine learning model being utilized.

6. Conclusion

In this study, we addressed critical challenges in SFP, particularly the issues of high-dimensional feature spaces and imbalanced datasets, which often degrade the performance of predictive models. To overcome these limitations, we proposed the application of the Zebra Optimization Algorithm as a feature selection method, aiming to enhance the accuracy and generalizability of SFP models. ZOA's global search capability and resilience against local optima make it well-suited for identifying the most relevant software metrics from complex datasets. The experimental evaluations were conducted on the BugHunter dataset, a large-scale and diverse benchmark that provides a realistic setting for assessing the effectiveness of SFP models. Our results demonstrate that feature selection using ZOA significantly improved model performance across multiple evaluation metrics. These findings confirm the potential of combining metaheuristic-based feature selection with deep learning architectures to develop robust and scalable SFP solutions. In future work, we will explore additional deep learning models and ensemble techniques while further enhancing model performance through advanced oversampling methods and hybrid FS strategies.

REFERENCES

[1] F. Bartumeus, M. G. E. da Luz, G. M. Viswanathan, and J. Catalan, "Animal search strategies: a quantitative random-walk analysis", *Ecology*, vol. 86, no. 11, pp. 3078–3087, 2005.

[2] Z. Dang and H. Wang, "Leveraging meta-heuristic algorithms for effective software fault prediction: a comprehensive study", *Journal of Engineering and Applied Science*, vol. 71, no. 1, p. 189, 2024.

[3] E. Trojovská, M. Dehghani, and P. Trojovský, "Zebra optimization algorithm: a new bio-inspired optimization algorithm for solving optimization problems", *IEEE Access*, vol. 10, pp. 49445–49473, 2022.

[4] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction", *Journal of Systems and Software*, vol. 169, p. 110691, 2020.

[5] M. Ali, T. Mazhar, T. Shahzad, Y. Y. Ghadi, S. M. Mohsin, S. M. A. Akber, and M. Ali, "Analysis of feature selection methods in software defect prediction models", *IEEE Access*, vol. 11, pp. 145954–145974, 2023.

[6] A. O. Balogun, S. Basri, L. F. Capretz, S. Mahamad, A. A. Imam, M. A. Almomani, V. E. Adeyemo, A. K. Alazzawi, A. O. Bajeh, and G. Kumar, "Software defect prediction using wrapper feature selection based on dynamic re-ranking strategy", *Symmetry*, vol. 13, no. 11, p. 2166, 2021.

[7] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, "Performance analysis of feature selection methods in software defect prediction: a search method approach", *Applied Sciences*, vol. 9, no. 13, p. 2764, 2019.

[8] A. M. Akbar, R. Herteno, S. W. Saputro, M. R. Faisal, and R. A. Nugroho, "Optimizing software defect prediction models: integrating hybrid grey wolf and particle swarm optimization for enhanced feature selection with popular gradient boosting algorithm", *Journal of Electronics, Electromedical Engineering, and Medical Informatics*, vol. 6, no. 2, pp. 169–181, 2024.

[9] R. Al-Wajih, S. J. Abdulkadir, N. Aziz, Q. Al-Tashi, and N. Talpur, "Hybrid binary grey wolf with harris hawks optimizer for feature selection", *IEEE Access*, vol. 9, pp. 31662–31677, 2021.

[10] O. Almomani, "A feature selection model for network intrusion detection system based on PSO, GWO, FFA and GA algorithms", *Symmetry*, vol. 12, no. 6, pp. 1046, 2020.

[11] N. M. Sallam, A. I. Saleh, H. A. Ali, and M. M. Abdelsalam, "An efficient strategy for blood diseases detection based on grey wolf optimization as feature selection and machine learning techniques", *Applied Sciences*, vol. 12, no. 21, p. 10760, 2022.

[12] R. B. Said, Z. Sabir, and I. Askerzade, "CNN-BiLSTM: a hybrid deep learning approach for network intrusion detection system in software-defined networking with hybrid feature selection", *IEEE Access*, vol. 11, pp. 138732–138747, 2023.

[13] M. Khan and R. Kishwar, "A novel software defect prediction model using two-phase grey wolf optimisation for feature selection", vol. 27, no. 9, pp. 12185-12207, 2024.

[14] R. Malhotra, S. Chawla, and A. Sharma, "Software defect prediction based on multi-filter wrapper feature selection and deep neural network with attention mechanism", *Neural Computing and Applications*, vol. 37, pp. 22621–22648, 2025.

[15] S. C. Rathi, S. Misra, R. Colomo-Palacios, R. Adarsh, L. B. M. Neti, and L. Kumar, "Empirical evaluation of the performance of data sampling and feature selection techniques for software fault prediction", *Expert Systems with Applications*, vol. 223, p. 119806, 2023.

[16] Y. Liu, "An improved zebra optimization algorithm", *International Journal of Engineering Research and Management (IJERM)*, vol. 12, no. 3, pp. 86–90, 2025.

[17] M. P. LaValley, "Logistic regression", *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.

[18] N. V. Thieu and S. Mirjalili, "Mealpy: An open-source library for latest meta-heuristic algorithms in Python", *Journal of Systems Architecture*, vol. 139, pp. 102871, 2023.

[19] S. Balasubramaniam and S. G. Gollagi, "Software defect prediction via optimal trained convolutional neural network", *Advances in Engineering Software*, vol. 169, p. 103138, 2022.

[20] E. Borandag, "Software fault prediction using an RNN-based deep learning approach and ensemble machine learning techniques", *Applied Sciences*, vol. 13, no. 3, p. 1639, 2023.

[21] M. R. Islam, M. Begum, and M. N. Akhtar, "Recursive approach for multiple step-ahead software fault prediction through long short-term memory (LSTM)", *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 25, no. 7, pp. 2129–2138, 2022.