Performance Analysis of Deep Learning Models for Software Fault Prediction Using the BugHunter Dataset

Dang Thi Kim Ngan, Dao Khanh Duy, Ha Thi Minh Phuong, Nguyen Thanh Binh
The University of Danang - Vietnam-Korea University of Information and Communication Technology, Danang, Vietnam

Correspondence: Nguyen Thanh Binh. Email: ntbinh@vku.udn.vn
Received: 30/01/2025, revised: 20/05/2025, accepted: 23/06/2025
DOI: 10.32913/mic-ict-research-vn.v2025.n1.1374

Abstract: Software fault prediction (SFP) involves the identification of potentially fault-prone modules before the testing phase in the software development lifecycle. By predicting faults early in the development process, the SFP process enables software developers to focus their efforts on components that may contain faults, thereby enhancing the overall quality and reliability of the software. Machine learning and deep learning techniques have been widely applied to train SFP models. However, these approaches face several challenges, including irrelevant or redundant features, imbalanced datasets, overfitting, and complex model structures. The NASA dataset from the PROMISE repository is the most commonly used dataset for fault prediction. Recently, the BugHunter dataset with its substantially larger number of instances was explored to train the SFP models. In this study, we present the comparative study of three deep learning models, including Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and four machine learning models as K-Nearest Neighbors (KNN), Multilayer Perceptron (MLP), Adaptive Boosting (AdaBoost), Extreme Gradient Boosting (XGB) to investigate the performance of SFP models on the BugHunter dataset. We employ the Lasso method for feature selection and apply the Synthetic Minority Oversampling Technique (SMOTE) to address the issue of imbalanced data, aiming to enhance the accuracy of the results. The experimental findings reveal that CNN and RNN outperformed other machine learning models, achieving the best overall performance.

Keywords: *software fault prediction, machine learning, BugHunter dataset.*

Tiêu đề: **Đánh giá các mô hình học sâu cho bài toán dự đoán lỗi phần mềm trên bộ dữ liệu BugHunter**

Tóm tắt: Dự đoán lỗi phần mềm (SFP) quá trình xác định các mô-đun có khả năng chứa lỗi trước giai đoạn kiểm thử trong vòng đời phát triển phần mềm. Việc dự đoán lỗi sớm trong quá trình phát triển cho phép các nhà phát triển phần mềm tập trung nguồn lực vào các thành phần tiềm ẩn lỗi, từ đó nâng cao chất lượng và độ tin cậy tổng thể của phần mềm. Các kỹ thuật học máy và học sâu đã được áp dụng rộng rãi để huấn luyện các mô hình SFP. Tuy nhiên, những phương pháp này vẫn đối mặt với nhiều thách thức, bao gồm sự tồn tại của các đặc trưng không liên quan hoặc dư thừa, sự mất cân bằng dữ liệu, hiện tượng quá khớp (overfitting), và cấu trúc mô hình phức tạp. Trong số các bộ dữ liệu được sử dụng cho nhiệm vụ dự đoán lỗi, bộ dữ liệu NASA từ kho PROMISE là bộ phổ biến nhất. Gần đây, bộ dữ liệu BugHunter với số lượng mẫu lớn hơn đáng kể đã được khai thác nhằm huấn luyện các mô hình SFP. Nghiên cứu này trình bày một phân tích so sánh giữa ba mô hình học sâu gồm Mạng Nơ-ron Tích chập (CNN), Mạng Nơ-ron Hồi tiếp (RNN), và Bộ nhớ dài-ngắn hạn (LSTM), cùng với bốn mô hình học máy bao gồm K-Nearest Neighbors (KNN), Multilayer Perceptron (MLP), Adaptive Boosting (AdaBoost), và Extreme Gradient Boosting (XGB) nhằm đánh giá hiệu năng của các mô hình SFP trên bộ dữ liệu BugHunter. Phương pháp Lasso được sử dụng để lựa chọn đặc trưng, trong khi kỹ thuật Synthetic Minority Oversampling Technique (SMOTE) được áp dụng nhằm giải quyết vấn đề mất cân bằng dữ liệu, với mục tiêu cải thiện độ chính xác của mô hình. Kết quả thực nghiệm cho thấy, CNN và RNN vượt trội hơn các mô hình học máy còn lại và đạt được hiệu suất tổng thể tốt nhất.

Từ khóa: *dự đoán lỗi phần mềm, học máy, tập dữ liệu BugHunter*

I. INTRODUCTION

Developers follow a structured set of steps to build reliable and high-quality software. Errors may occur throughout any phase of the development process. Software faults are emerging issues that produce unexpected results, potentially causing significant damage. Software testing is considered an essential phase within the software development life cycle (SDLC) [1], as it plays a vital role in ensuring the delivery of high-quality software. SFP identifies faults early in the SDLC, improving the final product's quality quickly and cost-effectively. SFP focuses on identifying modules likely to contain faults such as files, classes, methods, and functions based on defect data gathered from past software projects. Predicting potential faults early helps improve software quality by addressing critical faults before they affect functionality, reliability, or security. Many machine learning (ML) techniques have been investigated for constructing SFP models, yielding encouraging outcomes on publicly available datasets. Ensemble learning, along with deep learning models, has been extensively employed and has shown considerable success in this context [2, 3]. These techniques utilise fault records from previous software projects to develop models that anticipate faults in new systems [4]. However, the performance of fault prediction models is affected by various factors such as datasets, software metrics, machine learning techniques, feature selection methods, evaluation criteria, and issues related to datasets [5].

Recently, Ferenc *et al.* [6] introduced a new dataset called BugHunter, which features a substantially greater number of instances than the NASA datasets. This facilitates the ability of machine learning and deep learning models to improve their performance. One of the main challenges with the BugHunter dataset is the issue of imbalanced data, where the number of non-fault instances is much greater than the number of fault instances. By addressing this problem using oversampling techniques, both machine learning and deep learning models can achieve better performance. This also opens a new avenue for research, where researchers can apply and compare various deep learning architectures, algorithms, and techniques to push the boundaries of machine learning performance.

In this study, we present a comparative study of four machine learning techniques including KNN, MLP, XGBoost and AdaBoost and three deep learning models, namely CNN, RNN, and LSTM using the BugHunter Dataset [6] to compare their performance. The experimental result shows that RNN, CNN and LTSM achieved higher performance than machine learning models by 10.16%, 12.28%, 12.28%, and 5.61% in terms of accuracy, AUC, F1-score, and recall, respectively. In the next section, we present related work in SFP. The experimental design is illustrated in section II. Section III provides the experimental results. Our conclusion is presented in section IV.

II. RESEARCH METHODOLOGY

Researchers have extensively explored machine learning and deep learning (DL) classifiers using various datasets and performance metrics. This section reviews significant research works on SFP, highlights the rationale for selecting specific DL techniques, and discusses existing challenges in SFP.

1. Machine Learning Techniques

Numerous studies have demonstrated that machine learning techniques receive considerable focus in SFP because of their capability to analyse diverse software metrics and accurately forecast software faults. In [7], the authors presented a CRC-based software fault prediction approach. They compared the performance of various models, such as Naive Bayes, neural networks, and C4.5, by utilizing the NASA datasets. The results show that the proposed model outperforms the others. Ensemble methods were also investigated in [8], where the authors compared filter-based feature ranking techniques like Gain Ratio and Information Gain. They developed models using Naive Bayes (NB), MLP, KNN, SVM, and logistic regression (LR). The experiments demonstrated the effectiveness of ensemble approaches.

Rathore and Kumar [4] showed that most recent studies have used supervised learning techniques such as Decision Tree (DT), SVM, MLP, and ensemble learning as Random Forest (RF), followed by statistical techniques for building SFP models. In a systematic review conducted by Malhotra [9], C4.5, NB, MLP, SVM, and RF were identified as the machine learning techniques most commonly applied in SFP.

Cynthia *et al.* [10] concentrated on enhancing the effectiveness of machine learning techniques in SFP. They introduced a new feature transformation method that employs a genetic algorithm to identify the optimal representation of the data, which helps better distinguish between buggy and bug-free code. As a result, popular machine learning models such as Random Forest, KNN, LR, and NB achieved significantly higher performance than when using the original dataset.

Recently, Ferenc *et al.* [6] employed traditional machine learning models including Naive Bayes Multinomial, Logistic Regression, SGD, Simple Logistic, Voted Perceptron, Decision Table, OneR, J48 (C4.5), RF, and Random Tree to assess the performance of the new BugHunter dataset

in predicting software bugs. The results achieved high F-measure values. A study by Pandey *et al.* [5] compared the performance of various ML algorithms in SFP. The results showed that algorithms such as Random Forest, SVM, Naive Bayes, C4.5, Logistic Regression, and MLP were the top choices. These algorithms have proven to be capable of accurately predicting code segments at risk of errors, helping software developers proactively prevent potential errors.

2. Deep Learning Techniques

Deep Learning, which belongs to the broader area of machine learning, employs multilayer Neural Networks to process extensive datasets, identify underlying patterns, and transfer learned representations across various tasks [11]. By integrating supervised, unsupervised, hybrid, and reinforcement learning techniques, the approach detailed in [5] demonstrates high effectiveness in SFP for large datasets exceeding 10,000 instances, surpassing traditional methods in terms of predictive capability. Dam *et al.* [12] introduced a tree-structured LSTM model designed to automatically extract source code features for fault prediction. The model utilises the Abstract Syntax Tree (AST) structure of source code and functions through four main stages: converting the code into an AST, transforming the labels of AST nodes into vector embeddings, and applying a tree-structured LSTM network to generate a comprehensive vector representation of the full AST.

In [13], Liang *et al.* proposed an SFP model that combines word embedding and LSTM. The process involves three steps: extracting tokens from the AST, converting the tokens into vectors, and using these vectors with labels to build an LSTM. The LSTM learns the semantic meaning of the code to identify faults. Experiments conducted on eight open-source software projects demonstrate that the model achieves better performance than current fault prediction techniques. In contrast to earlier studies that concentrate on extracting features from tree-based structures, this model prioritises capturing the semantic meaning of programs. Phan and Le Nguyen [14] introduced a CNN-driven approach that characterises programs by generating Control Flow Graphs (CFGs) from assembly code to model execution behaviour. They then applied multiview, multilayer CNNs to analyse the CFG data for fault prediction.

Farid *et al.* [15] proposed the CBIL framework, leveraging CNN and Bi-LSTM to enhance SFP models, showing a 30% enhancement over baseline models and improving the F-measure by 25% compared to CNN. Similarly, an 1D-CNN model was introduced by Zain *et al.* [16]. This model demonstrates superior performance over traditional ML and CNN models in both accuracy and F1-score, confirming its effectiveness for fault proneness prediction. Meanwhile, Qiao *et al.* [17] introduced a DL-based model utilizing software metrics from real-world datasets, outperforming state-of-the-art methods with significantly higher fault prediction accuracy.

In general, researchers have investigated a range of traditional machine learning techniques and deep learning architectures to improve the effectiveness of SFP models, leading to notable improvements in accuracy, precision, recall, and other evaluation metrics. However, datasets remain the most critical factor influencing the quality of these models. Considering the BugHunter dataset [6], it contains significantly more instances compared to previous datasets, which facilitates the improvement of ML and DL performance over prior research. This study performs experiments on the BugHunter dataset utilising multiple techniques, with further details provided in section II.

3. Experiment Design

In this paper, we examine the performance of four conventional machine learning and ensemble boosting algorithms, namely KNN, MLP, XGBoost, and AdaBoost as well as three deep learning models: CNN, RNN, and LSTM. The experimental steps are presented in Fig. 1. In the first stage, we collected seven projects from BugHunter, with the details of each dataset described in Table I. The effectiveness of SFP models is highly influenced by the quality of the training data, making the pre-processing stage a critical factor in this context. The data was partitioned into training and testing sets for assessing the model's performance. Subsequently, Lasso regression combined with eight-fold cross-validation was employed to determine the best feature. Additionally, SMOTE was applied to balance the dataset. The balanced data was then used to train models using the parameters described in section II.7.

4. Datasets

Seven different Java projects extracted from the BugHunter dataset [6] were used in this comparative study. As shown in Table I, the number of non-faulty instances is greater than that of faulty instances, so the datasets are imbalanced. Each project includes independent attributes such as Comment Rules, Coupling Rules, etc. and the dependent attribute is number of bugs. Table I presents the description of the used projects.

5. Feature Selection

Least Absolute Shrinkage and Selection Operator (Lasso) [18], combined with five-fold cross-validation, was utilized to identify the most relevant features, thereby enhancing

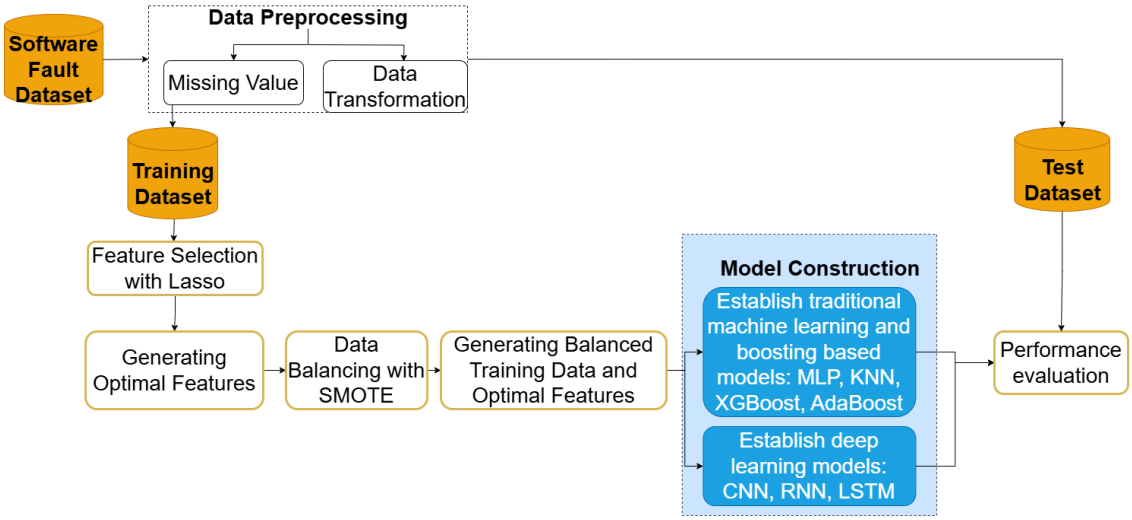

Figure 1. The main steps of the evaluation.

Table I
BUGHUNTER DATASETS USED IN THE STUDY

Project	Inst.	Faulty	Non-F.	Ratio (%)
Ceylon	2087	508	1579	23.34
Orxy	810	77	733	9.51
Titan	785	168	617	21.40
Orientdb	9445	2589	6856	27.41
Broadleaf	4709	1025	3684	21.77
MapDB	1456	480	976	32.97
Neo4j	7030	1841	5189	26.19

model learning and minimizing overfitting. The LASSO formula can be defined as follows:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left(\frac{1}{2N} \sum_{i=1}^{N} \left(y_i - X_{ip}\beta_p \right)^2 + \lambda \sum_{j=1}^{p} ||\beta_j|| \right) \quad (1)$$

where X_{ip} is pth predictor (feature), y_i is the value of the response and β_j is the regression coefficient of the pth feature.

6. Handling Imbalanced Data

The BugHunter dataset presents a challenge due to its imbalanced nature. Therefore, in this study, SMOTE [19] was utilized. To maintain the reliability and validity of our model, the dataset was split into training and testing subsets, applying SMOTE solely to the training portion instead of the full dataset.

7. Parameter Configuration

In this study, the hyperparameter optimisation Optuna [20] was utilised for tuning hyperparameters in each machine learning model. Furthermore, we applied 10-fold cross-validation to assess the performance of the comparative methods. During each iteration, the model was trained on eight segments of the dataset, with the remaining segment set aside for evaluation. Tables II-IV describe some parameters and their values in deep learning models (CNN, RNN, LSTM).

8. Machine Learning and Deep Learning algorithms

As presented in sections II.1 and II.2, researchers have explored and applied various ML and DL approaches to improve the effectiveness of SFP models. Among the numerous options, we selected the following algorithms for this comparative study.

1) KNN represents a fundamental classification algorithm, particularly suitable in scenarios where limited or no prior knowledge regarding the data distribution is available [21]. The algorithm assigns a class label to a query instance by analyzing the labels of its K nearest neighbors in the feature space. Specifically, the predicted label corresponds to the class that appears most frequently among the K nearest neighbors.
2) MLP belongs to the family of Artificial Neural Networks (ANN) and features several layers of neurons arranged sequentially. This architecture aims to

Table II
PARAMETER CONFIGURATION FOR CNN

Parameter	Definition	Value
layer_1_filters	Number of filters in first convolutional layer	50
layer_2_filters	Number of filters in second convolutional layer	20
kernel_size	Size of the convolutional kernel	3
pool_size	Size of max pooling window	2
dense_units	Number of units in hidden dense layer	64
batch_size	Number of samples per gradient update	32
epochs	Number of complete passes through the training dataset	200
learning_rate	Optimizer learning rate	0.01
optimizer	Optimization algorithm	Adam
loss_function	Loss function for model training	Binary Cross-entropy
activation_function	Activation function in hidden layers	ReLU
output_activation	Activation function in output layer	Sigmoid

Table III
PARAMETER CONFIGURATION FOR RNN

Parameter	Definition	Value
rnn_units	Number of units in RNN layer	32
dense_1_units	Number of units in first dense layer	32
batch_size	Number of samples per gradient update	32
epochs	Number of complete passes through the training dataset	200
learning_rate	Optimizer learning rate	0.01
optimizer	Optimization algorithm	Adam
loss_function	Loss function for model training	Binary Cross-entropy
activation_rnn	Activation function in RNN layer	ReLU
activation_dense	Activation function in dense layer	ReLU
output_activation	Activation function in output layer	Sigmoid

Table IV
PARAMETER CONFIGURATION FOR LSTM

Parameter	Definition	Value
layer_1_dim	Number of units in first LSTM layer	128
layer_2_dim	Number of units in second LSTM layer	64
batch_size	Number of samples per gradient update	64
epochs	Number of complete passes through the training dataset	200
learning_rate	Optimizer learning rate	0.001
dropout_rate	Dropout probability for regularization	0.2
l2_regularization	L2 regularization coefficient	0.01
optimizer	Optimization algorithm	Adam
loss_function	Loss function for model training	Binary Cross-entropy
output_activation	Activation function in output layer	Sigmoid

replicate the neural processing functions observed in the human brain. The MLP generally comprises an input layer, multiple hidden layers, and an output layer. The hidden layers which are unobservable from outside the network [22]-are essential for capturing and learning intricate patterns within the data.

3) AdaBoost [23] is a member of the Boosting family of algorithms. This method operates iteratively by prioritizing misclassified samples during training, adjusting the sample distribution accordingly, and continuing this process until the weak classifier has been trained a predefined number of times.
4) XGB [24] is also a part of the Boosting family of algorithms. The aim of XGB is to minimize the loss function by constructing decision trees sequentially, where each new tree corrects the errors made by the previous trees. XGB enables regularization (L1 and L2), parallel processing, and automatic handling of missing data.
5) CNN is a prominent deep learning framework inspired by the visual perception systems found in biological entities [25]. Its architecture commonly includes essential layers such as convolutional, activation, pooling, fully connected, dropout, and batch normalization layers.
6) RNN is a neural network architecture specifically developed for handling sequential data. Its strength lies in a high-dimensional hidden state governed by nonlinear dynamics, allowing it to retain and utilise past information effectively [26].
7) LSTM was introduced to address the exploding/vanishing gradient problems that typically arise when

learning long-term dependencies [27] by using memory cells and gating mechanisms to retain or forget information over long sequences selectively.

9. Performance Metrics

In order to evaluate and deeply understand the performance of the models, we examined them based 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 percentage of correctly predicted positive samples out of the total predicted positive samples. It is represented as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Accuracy is the ratio of correct predictions to the total instances in the target attribute. The value of accuracy ranges from 0 to 1.

F1-score is calculated as:

$$F1\text{-}score = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

The F1-score increases when both precision and recall are high, providing a balanced measure of a model's performance, especially in handling imbalanced datasets.
Recall measures the percentage of correctly predicted positive samples out of all actual positive samples. It is represented as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

The AUC (Area Under the Curve) is used to evaluate the efficiency of the SFP model. A curve that approaches the upper-left corner of the plot indicates good model performance, whereas a curve that deviates significantly from this area suggests poorer performance.

III. EXPERIMENT RESULTS

This section presents the experimental findings from a comparative analysis of machine learning and deep learning models across various projects within the BugHunter dataset, evaluated using Accuracy, AUC, F1-score, and Recall metrics.

As shown in Table V, CNN, RNN, and LSTM demonstrated superior performance in most projects. In the Ceylon-ide-eclipse project, CNN achieved the highest performance in all metrics, achieving an value of 63.4%, an AUC value 67.8%, an F1-score value 62.7%, and recall value 64.3%. Meanwhile, RNN demonstrated better performance compared to traditional machine learning models, achieving an accuracy value 62%, an AUC value 66.4%, F1-score value 61.8%, and recall value 63.9%. LSTM outperformed traditional machine learning models only in terms of AUC, with a value value 63.2%. RNN showed exceptional performance on the Orxy project, achieving the highest accuracy (91.0%) and recall (96.2%) values while CNN achieved the best AUC (95.0%).

For the Titan project, the RNN model consistently outperformed all other models by significant margins, achieving the highest performance across all evaluated metrics, including accuracy (65.6%), AUC (73.5%), F1-score (63.2%), and recall (62.0%). Similarly, in the Neo4j project, the CNN model demonstrated superior performance across all metrics, attaining an accuracy of 56.0%, an AUC of 57.8%, an F1-score of 59.5%, and a recall of 65.3%.

In the MapDB project, deep learning models demonstrated superior performance across multiple metrics. The CNN achieved the highest accuracy value of 62.5%, the RNN attained the highest AUC value of 66.2%, while the LSTM exhibited strong performance in terms of an F1-score value of 68.0% and a recall value of 90.7%.

On the other hand, traditional machine learning models achieved better performance than deep learning models in some cases. For instance, in the Broadleaf-Commerce project, KNN achieved superior performance across most metrics, with an accuracy value of 68.6%, an F1-score value of 68.4%, and a recall value of 69.2%. The highest AUC, however, was attained by the RNN model, with a value of 75.6%. Similarly, in the OrientDB project, the MLP model demonstrated robust performance, achieving the highest F1-score and recall values of 59.5% and 59.5%, while the RNN model achieved the best AUC value of 65.4%.

Moreover, drawing upon the study by Ferenc *et al.* [6], we compare their best performing techniques with our experimental results in terms of F1-score as summarized in Table VI. For the MapDB project, RNN slightly outperforms CNN, with F1-score values of 0.643 and 0.604, respectively. Meanwhile, Random Forest demonstrates competitive performance, achieving the highest F1-score value of 0.736 on the BroadleafCommerce project, while also performing well on Orientdb (0.623). Lastly, LSTM showed varied performance, with its best score of 0.832 on the Oryx project. Overall, deep learning models (CNN, RNN and LSTM) achieved better performance compared with the

Table V
PERFORMANCE COMPARISON OF MACHINE LEARNING AND DEEP LEARNING MODELS

Project	Metric	KNN	MLP	AdaBoost	XGB	CNN	RNN	LSTM
Ceylon-ide-eclipse	Accuracy	0.538	0.544	0.571	0.583	**0.634**	0.620	0.581
	AUC	0.543	0.582	0.568	0.587	**0.678**	0.664	0.632
	F1-score	0.530	0.537	0.562	0.578	**0.627**	0.618	0.562
	Recall	0.541	0.547	0.577	0.586	**0.643**	0.639	0.548
Orxy	Accuracy	0.688	0.713	0.650	0.613	0.908	**0.910**	0.834
	AUC	0.750	0.757	0.770	0.725	**0.950**	0.945	0.905
	F1-score	0.669	0.712	0.621	0.544	0.911	**0.914**	0.832
	Recall	0.742	0.712	0.720	0.782	0.954	**0.962**	0.832
Titan	Accuracy	0.494	0.530	0.500	0.530	0.617	**0.656**	0.570
	AUC	0.513	0.500	0.493	0.500	0.680	**0.735**	0.613
	F1-score	0.488	0.528	0.474	0.528	0.573	**0.632**	0.532
	Recall	0.495	0.532	0.500	0.531	0.558	**0.620**	0.536
Orientdb	Accuracy	0.576	0.594	0.592	0.580	0.600	0.602	**0.616**
	AUC	0.599	0.634	0.630	0.628	0.641	**0.654**	0.650
	F1-score	0.571	**0.595**	0.592	0.575	0.565	0.557	0.576
	Recall	0.579	**0.595**	0.592	0.584	0.524	0.505	0.522
Broadleaf-Commerce	Accuracy	**0.686**	0.673	0.650	0.672	0.639	0.683	0.623
	AUC	0.725	0.732	0.709	0.748	0.702	**0.756**	0.693
	F1-score	**0.684**	0.667	0.646	0.665	0.584	0.680	0.584
	Recall	**0.692**	0.688	0.657	0.688	0.513	0.676	0.534
MapDB	Accuracy	0.586	0.603	0.588	0.584	**0.625**	0.615	0.575
	AUC	0.622	0.632	0.619	0.609	0.655	**0.662**	0.620
	F1-score	0.560	0.592	0.586	0.582	0.604	0.643	**0.680**
	Recall	0.614	0.614	0.690	0.586	0.584	0.706	**0.907**

Table VI
PERFORMANCE COMPARISON OF DEEP LEARNING MODELS WITH BEST-PERFORMING TECHNIQUES IN [8]

Project	Random Forest	SimpleLogistic	SGD	CNN	RNN	LTSM
Ceylon-ide-eclipse	0.539	–	–	**0.627**	0.618	0.562
Oryx	0.667	–	–	0.911	**0.914**	0.832
Orientdb	**0.623**	–	–	0.565	0.557	0.576
MapDB	–	0.561	–	0.604	0.643	**0.680**
BroadleafCommerce	**0.736**	–	–	0.584	0.680	0.584
Titan	–	–	0.579	0.573	**0.632**	0.532

ML models on 5 out of 7 projects in terms of F1-score.

The results imply that RNN and CNN models demonstrate reliable effectiveness, confirming their relevance to this domain. However, traditional machine learning approaches remain viable alternatives in specific scenarios, demonstrating competitive performance in certain metrics.

IV. CONCLUSIONS

Choosing models in SFP plays a critical role in determining overall performance. Although machine learning models have shown considerable effectiveness, deep learning approaches possess the capability to address the inherent limitations of traditional machine learning techniques. Our objective is to evaluate and contrast the performance of various machine learning and deep learning models on the BugHunter dataset, using Accuracy, F1-score, AUC, and Recall as evaluation metrics. In general, deep learning models demonstrated their superiority by achieving higher performance than machine learning models on most projects. In particular, RNN, CNN, and LSTM outperformed traditional machine learning models, yielding respective improvements of 10.16%, 12.28%, 12.28%, and 5.61% in Accuracy, AUC, F1-score, and Recall. In the future, we plan to evaluate more state-of-the-art deep learning models on the BugHunter dataset and improve the performance of SFP models by handling imbalanced data and irrelevant or redundant feature issues. We expect these models to deliver even higher performance, providing significant advantages in the early prediction of software faults.

REFERENCES

[1] S. Najihi, S. Elhadi, R. A. Abdelouahid, and A. Marzak, "Software testing from an agile and traditional view," *Procedia Computer Science*, vol. 203, pp. 775–782, 2022.

[2] A. Balaram and S. Vasundra, "Prediction of software fault-prone classes using ensemble random forest with adaptive synthetic sampling algorithm," *Automated Software Engineering*, vol. 29, no. 1, p. 6, 2022.

[3] M. Mangla, N. Sharma, and S. N. Mohanty, "A sequential ensemble model for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 18, no. 2, pp. 301–308, 2022.

[4] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, pp. 255–327, 2019.

[5] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications*, vol. 172, p. 114595, 2021.

[6] 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.

[7] C. Jin, S.-W. Jin, and J.-M. Ye, "Artificial neural network-based metric selection for software fault-prone prediction model," *IET software*, vol. 6, no. 6, pp. 479–487, 2012.

[8] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, 2018.

[9] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[10] S. T. Cynthia, B. Roy, and D. Mondal, "Feature transformation for improved software bug detection models," in *Proceedings of the 15th Innovations in Software Engineering Conference*, pp. 1–10, 2022.

[11] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications surveys & tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[12] H. K. Dam, T. Pham, S. W. Ng, T. Tran, J. Grundy, A. Ghose, T. Kim, and C.-J. Kim, "A deep tree-based model for software defect prediction," *arXiv preprint arXiv:1802.00921*, 2018.

[13] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic lstm model for software defect prediction," *IEEE Access*, vol. 7, pp. 83812–83824, 2019.

[14] A. V. Phan and M. Le Nguyen, "Convolutional neural networks on assembly code for predicting software defects," in *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pp. 37–42, IEEE, 2017.

[15] A. B. Farid, E. M. Fathy, A. S. Eldin, and L. A. Abd-Elmegid, "Software defect prediction using hybrid model (cbil) of convolutional neural network (cnn) and bidirectional long short-term memory (bi-lstm)," *PeerJ Computer Science*, vol. 7, p. e739, 2021.

[16] Z. M. Zain, S. Sakri, N. H. Asmak Ismail, and R. M. Parizi, "Software defect prediction harnessing on multi 1-dimensional convolutional neural network structure.," *Computers, Materials & Continua*, vol. 71, no. 1, 2022.

[17] L. Qiao, X. Li, Q. Umer, and P. Guo, "Deep learning based software defect prediction," *Neurocomputing*, vol. 385, pp. 100–110, 2020.

[18] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[21] S. B. Imandoust, M. Bolandraftar, *et al.*, "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," *International journal of engineering research and applications*, vol. 3, no. 5, pp. 605–610, 2013.

[22] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009.

[23] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[24] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[25] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[26] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 1017–1024, 2011.

[27] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5929–5955, 2020.

Dang Thi Kim Ngan obtained a Bachelor degree in Information Management at the University of Danang -University of Economics in 2011. She got an MSc degree of at the Chinese Culture University, Taiwan in 2014. She is currently serving as a lecturer at the University of Danang - Vietnam-Korea University of Information and Communication Technology. Her research focuses on software engineering.
Email: dtkngan@vku.udn.vn.

Dao Khanh Duy is currently pursuing a Bachelor of Engineering at the University of Danang - Vietnam-Korea University of Information and Communication Technology. He has been gaining valuable hands-on experience in research and development through internships at the King Mongkut's University of Technology North Bangkok and MGM technology partners. His current research focuses on software testing and software effort estimation.
Email: duydk.22git@vku.udn.vn.

Ha Thi Minh Phuong received a Bachelor degree in Information Technology from the University of Danang-University of Science and Technology in 2010. She earned M.Sc. degree in Computer Science at Yuan Ze University, Taiwan in 2013. She is a lecturer at the University of Danang - Vietnam-Korea University of Information and Communication Technology. She is currently pursuing the Ph.D. degree in Computer Science at the University of Danang. Her current research focuses on software testing, deep learning.
Email: htmphuong@vku.udn.vn.

Nguyen Thanh Binh graduated in Information Technology from the University of Danang - University of Science and Technology in 1997. He received a PhD. degree in Information Technology at Grenoble Institute of Technology, France in 2004. He has been qualified as Associate Professor since 2013. He is currently working at the University of Danang - Vietnam-Korea University of Information and Communication Technology. His research interests include software engineering and software quality.
Email: ntbinh@vku.udn.vn.