Improving Cross-Project Java Vulnerability Prediction Using Genetic Algorithm-Based Feature Optimization and Random Forest Classification

[1]Michael A. Olatunji, [2]Paul O. Adebayo, [3]Akeem A. Alabi, [4]Samuel O. Oluokun, [5]Olabanke O. Williams
[1]Department of Cybersecurity, Redeemer's University, Ede, Osun State, Nigeria
[1,5]Department of Computer Science, University of Ilorin, Ilorin, Kwara State, Nigeria
[3]Department of Information Systems, Osun State University, Osogbo, Osun State, Nigeria
[4]Department of Information Science and Technology, Kings University, Odeomu, Osun State, Nigeria
mikeolaus@gmail.com|adebayo.po@unilorin.edu.ng|akeem.alabi@uniosun.edu.ng|
so.oluokun@kingsuniversity.edu.ng|olabanke@unilorin.edu.ng

Received: 17-APRIL-2026; Reviewed: 25-JUNE-2026; Accepted: 25-SEPT-2026
https://dx.doi.org/10.46792/fuoyejet.v11i3.1565

ORIGINAL RESEARCH

Abstract— Software vulnerabilities in Java applications remain a critical security concern due to increasing system complexity and the limitations of traditional static analysis tools, which often generate high false-positive rates. Although machine learning-based Vulnerability Prediction Models (VPMs) have improved vulnerability detection, they continue to face challenges related to generalization and robustness, particularly in cross-project settings. This study proposes a Java vulnerability prediction framework that combines optimized feature selection with Random Forest (RF) classification. The key contributions of this study are twofold: (i) the construction of a new vulnerability prediction dataset that integrates traditional software metrics with vulnerability-oriented features extracted from static analysis tools, thereby providing a richer feature representation than conventional benchmark datasets, and (ii) a modified Genetic Algorithm incorporating an Extreme Two Genes (ETG) crossover operator and correlation-based fitness evaluation for enhanced feature optimization. Random oversampling is employed to address class imbalance. Experiments were conducted on Apache Tomcat 9.0.0.M1 (372 instances) and Eclipse Jetty 9.4.0.RC1 (324 instances). The proposed framework achieved within-project performance of 94% precision, 95% recall, and 95% F1-score for Tomcat, and 95% precision, 98% recall, and 95% F1-score for Jetty. Cross-project evaluation produced F1-scores of 76.9% and 70.9%. The results demonstrate that the proposed ETG-enhanced feature optimization framework improves vulnerability prediction performance, achieving strong within-project results while maintaining useful predictive capability across projects. Although cross-project prediction remains challenging and performance decreases relative to within-project prediction, the framework demonstrates robustness under cross-project variability and provides a foundation for more generalizable vulnerability prediction in Java software systems.

Keywords—Cross-project, Correlation, Model, Metrics, Vulnerability

1 INTRODUCTION

Java remains one of the most widely adopted programming languages for enterprise and large-scale applications due to its platform independence, object-oriented design, and strong ecosystem support (Sultana, Anu & Chong, 2024). Despite built-in safeguards such as automatic memory management and structured exception handling, Java applications continue to exhibit significant security vulnerabilities arising from coding errors, insecure API usage, weak authentication controls, and misconfigurations. These vulnerabilities may result in unauthorized access, data breaches, and system compromise, posing substantial risks to software reliability and security (Khare, Dutta, Li, Solko-Breslin, Alur & Naik, 2025; Malumfashi, Momoh, Muhammad & Dagaci, 2025). The increasing complexity of modern software systems has intensified the need for effective vulnerability detection mechanisms. Traditional static analysis tools are widely used to identify potential security flaws; however, they often suffer from high false-positive rates and limited contextual understanding (Wang, Tang, Tan, Wang, Liu, Fang, Xia & Zheng, 2024). Consequently, developers may spend considerable effort investigating non-critical warnings while actual vulnerabilities remain undetected (Olatunji, Ujah-Ogbuagu, Adebayo & Agbolade, 2026). To address these limitations, machine learning-based Vulnerability Prediction Models (VPMs) have been proposed to learn vulnerability patterns from historical software data using software metrics and code representations (Gunda, 2025; Zhao, Zhu & Peng, 2024).

Recent studies have increasingly employed deep learning and hybrid feature extraction techniques to capture both structural and semantic characteristics of source code. These approaches utilize representations such as Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and dependency graphs to model complex relationships within software systems, thereby improving the detection of subtle and context-dependent vulnerabilities (Zhao et al., 2024; Hussain, Nadeem,

Corresponding Author
Section B- ELECTRICAL/COMPUTER ENGINEERING & RELATED SCIENCES
Can be cited as:
Olatunji M. A., Adebayo P. O., Alabi A. A., Oluokun S. O., and Williams O. O. (2026). Improving Cross-Project Java Vulnerability Prediction Using Genetic Algorithm-Based Feature Optimization and Random Forest Classification. FUOYE Journal of Engineering and Technology (FUOYE**JET**), **11(1), 454-462.** https://dx.doi.org/10.46792/fuoyejet.v11i3.1565

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

Baber, Hamdi, Rajab, Al Reshan & Shaikh, 2024). By combining structural information with learned semantic embeddings, these models have demonstrated strong predictive performance, particularly in within-project settings. However, despite these advances, significant challenges remain, including limited cross-project generalizability, high computational overhead, and reduced robustness when applied to real-world and noisy datasets (Nguyen et al., 2025; Mächtle et al., 2025). Furthermore, many existing studies rely predominantly on traditional software metrics or learned code representations, while the integration of security-specific indicators generated by static analysis tools remains relatively underexplored (Dam, Tran, Pham, Ng, Grundy & Ghose, 2017; Zhao et al., 2024). Although feature selection techniques have been widely adopted to reduce dimensionality and improve predictive performance, comparatively less attention has been given to feature optimization approaches that explicitly account for both feature relevance and redundancy (Bassi & Singh, 2024; Manjula & Florence, 2018). As a result, opportunities remain to improve the quality of feature representations and enhance the robustness and generalizability of vulnerability prediction models, particularly in cross-project environments (Nguyen et al., 2025).

The objective of this study is to develop an effective Java vulnerability prediction framework that improves vulnerability detection performance through enhanced feature representation and feature optimization. The main contributions of the study are as follows:

1. Constructs a vulnerability prediction dataset that integrates traditional software metrics with security-specific features extracted using Automatic Static Analysis (ASA) tools, providing a richer representation of software vulnerability characteristics.
2. Proposes a modified Genetic Algorithm incorporating an Extreme Two Genes (ETG) crossover operator and correlation-based fitness evaluation to improve feature optimization.
3. Develops a Random Forest-based vulnerability prediction model using the optimized feature subset.
4. Evaluates the proposed framework under within-project and cross-project settings to assess its predictive performance, robustness, and generalizability.

The proposed framework integrates security-oriented features and optimized feature selection to improve Java vulnerability prediction, with emphasis on model robustness and cross-project applicability.

2 LITERATURE REVIEW

Early studies on vulnerability prediction leveraged machine learning and search-based techniques to improve detection accuracy. For instance, Tran and Cheng (2018) applied Long Short-Term Memory (LSTM) to learn semantic and syntactic code features, achieving strong within-project performance but poor cross-project generalization. Similarly, Sukanya and Saraswathy (2017) improved Genetic Algorithm (GA)-based defect prediction using an insertion operator, though their work focused mainly on defect classification without addressing class imbalance or feature diversity. Osman, Ghafari, Nierstrasz and Lungu (2017) demonstrated that combining change and source code metrics with Random Forest (RF) improves prediction performance but did not explore optimization-based enhancements. Manjula and Florence (2018) further enhanced Genetic Algorithm (GA) using modified chromosome representations and

Decision Trees, though challenges such as multicollinearity and cross-project prediction remained unaddressed. Collectively, these studies demonstrate that early machine learning and evolutionary approaches improved prediction accuracy but struggled to balance generalizability, feature quality, and dataset-specific challenges, limiting their effectiveness across diverse software projects. Recent advancements emphasize deep learning and hybrid approaches for improved feature representation. Hussain et al. (2024) proposed a hybrid model combining graph-based representations with CodeBERT and a Quantum Convolutional Neural Network (QCNN), achieving high predictive performance but with significant computational complexity. Similarly, Bajantri (2025) developed a Convolutional Neural Network–Recurrent Neural Network–Long Short-Term Memory (CNN–RNN–LSTM) model with transfer learning, demonstrating strong results but requiring large datasets and showing limited generalizability across programming languages. Compared with traditional machine learning approaches, deep learning and hybrid models provide richer semantic and structural representations of source code, enabling the detection of more complex vulnerability patterns. However, these benefits are often achieved at the expense of higher computational costs, reduced interpretability, and increased dependence on large training datasets.

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

The emergence of Large Language Models (LLMs) has further influenced vulnerability detection. Khare et al. (2025) evaluated models such as GPT-4 and CodeLlama, reporting moderate performance with improvements through advanced prompting strategies. However, LLMs struggled with complex vulnerabilities and exhibited limited consistency in real-world scenarios. Other deep learning approaches, such as Zhao et al. (2024), employed attention-based BiLSTM models using abstract syntax trees and dependency graphs, achieving improved detection performance but still facing limitations with certain vulnerability types and static analysis constraints. These findings indicate that although LLMs and advanced neural architectures can capture complex code semantics, their practical deployment remains constrained by issues of robustness, reproducibility, and computational cost. Recent studies have also explored scalability and robustness. Mächtle et al. (2025) introduced Trace Gadgets to reduce irrelevant code context, improving efficiency but limiting applicability to specific vulnerability classes. Nguyen et al. (2025) proposed VulGuard, achieving strong results in controlled settings but demonstrating significantly reduced performance in real-world environments due to noisy data.

Overall, the literature reveals a shift from traditional machine learning and evolutionary techniques toward deep learning, hybrid architectures, and LLM-based approaches. However, despite increasing model sophistication, persistent challenges such as cross-project generalization, class imbalance, feature redundancy, and robustness in noisy environments remain inadequately addressed. Machine learning and evolutionary approaches primarily emphasize feature engineering and model interpretability. Deep learning, hybrid models, and LLM-based techniques focus on automated feature extraction and semantic code understanding, resulting in different trade-offs between accuracy, explainability, and computational cost. Furthermore, relatively limited attention has been given to integrating security-specific indicators derived from static analysis tools with traditional software metrics for vulnerability prediction (Scandariato, Walden, Hovsepyan & Joosen, 2014; Wang et al., 2024). These observations motivate the need for a framework that integrates security-specific indicators with traditional software metrics while addressing feature redundancy, class imbalance, and cross-project robustness. The proposed study addresses these challenges by modifying the crossover phase of the Genetic Algorithm through the proposed Extreme Two Genes (ETG) crossover operator for feature optimization, followed by random oversampling and Random Forest-based vulnerability prediction.

3 METHODOLOGY

In this section, details of the techniques employed in this paper were presented. Modification strategy, datasets processing and stages involved in the work were discussed. Figure 1 depicts the study's framework for software security VPM for Java systems.

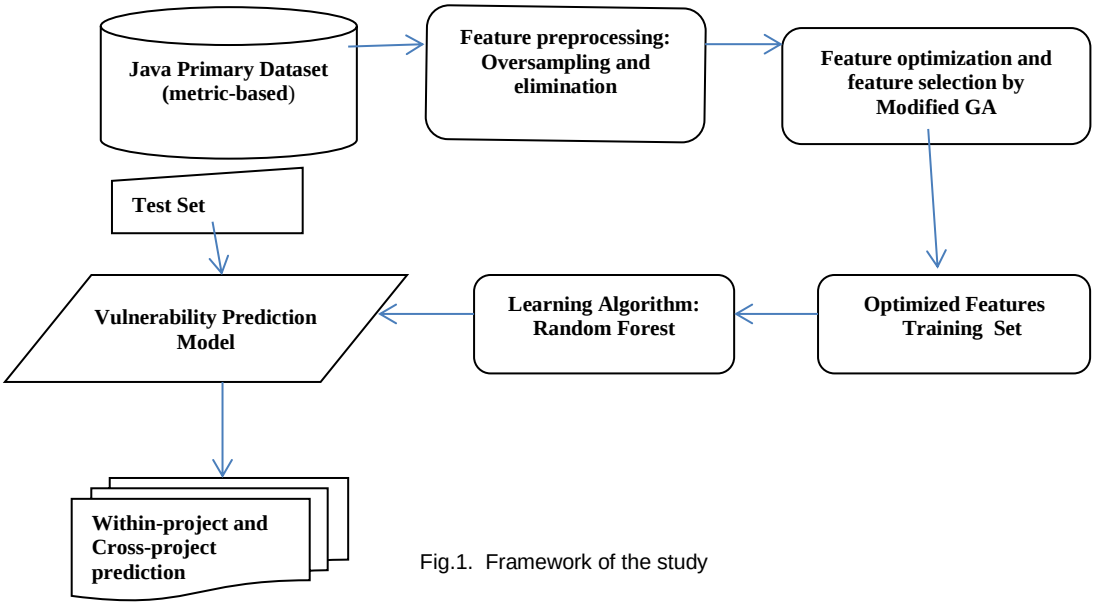

Fig.1. Framework of the study

3.1 Identification and Collection of Vulnerable Software System Samples: Dataset Projects

The software projects considered in this modeling research as datasets were identified and selected based on the following criteria: 1. The projects are open source projects to avoid licensing restrictions. 2. Projects with high number of reported vulnerabilities on Common Vulnerabilities and Exposures (CVE)

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

database. 3. Consideration of vulnerable files in the plug-ins and third-party components. Using these criteria, two Java projects (Tomcat 9.0.0. M1 and Jetty 9.4.0.RC1) presented in Table 1 were identified. Tomcat is both a web server and web container. Jetty is web server and Servlet container developed in Java. Tomcat 9.0.0. M1 and Jetty 9.4.0.RC1 project compressed zip folders were collected from GitHub.com. Tomcat 9.0.0. M1 extracted Java source code files are 372 while Jetty 9.4.0.RC1 extracted Java source code files are 324.

Table 1. Description of Java Dataset

Dataset	Metrics	Files	Vulnerable Files	Vulnerability %
Tomcat 9.0.0.M1	13	372	88	23.66%
Jetty 9.4.0.RC1	13	324	63	19.44%

3.2 Feature Creation and Preprocessing

Java source files were analyzed using three tools: Code Line Counter (CLC), Resource Standard Metrics (RSM), and Yet Another Source Code Analyzer (Yasca). CLC was used to extract source code and comment line metrics, while RSM computed cyclomatic and interface complexity at the file level. Yasca was employed to identify security-related issues, including cross-site scripting, SQL injection, weak authentication, cryptographic misuse, sensitive logging, poor coding practices, process control flaws, arbitrary file disclosure, and denial-of-service vulnerabilities. Files were labeled using CVE records, security advisories and developer resources. The resulting feature set comprised both traditional software metrics and security-specific indicators. All preprocessing and model evaluation tasks were conducted using WEKA 3.8. To address class imbalance, random oversampling with replacement was performed using the Resample class in WEKA's supervised instance filters. Since the number of vulnerable files was lower than the number of non-vulnerable files, minority-class instances were randomly duplicated to achieve a more balanced class distribution. Because the datasets were only moderately imbalanced, the oversampling procedure was applied once prior to feature optimization and model training. Feature selection and optimization were performed using a customized implementation of the WEKA Genetic Algorithm, modified in Java using the NetBeans IDE to incorporate proposed Extreme Two Genes (ETG) crossover operator. The modified algorithm was subsequently integrated into a local WEKA 3.8 environment for feature optimization. Prior to optimization, the Correlation Attribute Evaluator with the Ranker search method was used to retain the 10 most relevant features. This number was selected to balance information retention and dimensionality reduction, as preliminary experimentation indicated that including additional lower-ranked features did not improve and sometimes reduced cross-project prediction performance.

3.3 MODIFIED GENETIC ALGORITHM-BASED FEATURE SELECTION AND OPTIMIZATION

To improve the predictive capability of the vulnerability prediction model, a modified Genetic Algorithm (GA) was employed to identify an optimal subset of software features prior to classification. In the proposed representation, each chromosome corresponds to a software file (instance) in the dataset and consists of a set of software features (software metrics), while each gene represents an individual software feature (metric). Feature optimization was achieved by modifying the crossover operator of the Genetic Algorithm through the proposed Extreme Two Genes (ETG) crossover technique. Candidate feature subsets were evaluated using the Correlation-based Feature Selection (CfsSubsetEval) fitness function, which employs the Pearson correlation coefficient to maximize the relevance of the selected features to the vulnerability class while minimizing redundancy among them. The optimized feature subset was subsequently used to train the Random Forest classifier. The overall optimization procedure is summarized in Algorithm 1.

Algorithm 1: Modified Genetic Algorithm for Feature Selection

Input: Training dataset

Output: Optimized feature subset and optimized training dataset

1. Initialize population size, crossover probability, mutation probability, maximum number of generations, and fitness threshold.
2. Generate an initial population of binary chromosomes, where each chromosome represents a candidate feature subset.
3. For each chromosome X, compute its fitness using the Pearson correlation coefficient with the vulnerability status Y, as defined in Equation (1):

$$\text{Correl}(X, Y) = \frac{\Sigma (x - \bar{x})(y - \bar{y})}{\sqrt{\Sigma(x - \bar{x})^2 \, \Sigma(y - \bar{y})^2}} \quad (1)$$

Where:

x is the value of the selected feature for a software instance.

y is the corresponding vulnerability class label (vulnerable or non-vulnerable) for the software instance.

$\bar{x}$ is the mean value of feature X. $\bar{y}$ is the mean value of the vulnerability class Y.

$\sum$ denotes the summation over all software instances in the training dataset.

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

Correl (X,Y) is the Pearson correlation coefficient between feature X and the vulnerability class Y.

4. Combine the correlation score with feature frequency obtained from 10-fold cross-validation.
   a. Select fitter chromosomes to form the mating pool.
   b. Apply the Extreme Two Genes (ETG) crossover operator by exchanging the first two and last two genes between selected parent chromosomes.
   c. Apply mutation by randomly flipping binary gene values according to the mutation probability.
   d. Evaluate offspring fitness and update the population.
7. End While
8. Return the best-performing chromosome, optimized feature subset, and optimized training dataset.

The modified GA was executed for 100 generations using stratified 10-fold cross-validation to ensure reliable model evaluation. Selection was performed with replacement to preserve highly fit chromosomes across generations. The proposed Extreme Two Genes (ETG) crossover operator exchanges the first two and last two genes between parent chromosomes to promote diversity while reducing the likelihood of premature convergence. Mutation was implemented by randomly flipping binary gene values according to the predefined mutation probability. During fitness evaluation, feature subsets with high relevance to the vulnerability class and low inter-feature redundancy were preferred, thereby reducing overfitting and improving classifier generalization. For reproducibility, the proposed Extreme Two Genes (ETG) crossover operator was implemented in Java using the NetBeans IDE by modifying the Genetic Algorithm (GeneticSearch) in WEKA 3.8. Specifically, the default crossover operator was replaced with the proposed ETG crossover operator, while all other components remained unchanged. The modified Genetic Algorithm implementation was then used for feature selection and optimization. The optimization process employed a population size of 372 chromosomes, a crossover probability of 0.6, a mutation probability of 0.033, a maximum of 100 generations, and a random seed of 1. Feature ranking was performed using CorrelationAttributeEval with the Ranker search method, retaining the top 10 features. These parameter settings were kept constant throughout all experiments.

3.4 Data Splitting and Classification

The dataset was split into training (80%) and testing (20%) sets for within-project prediction. For cross

5. Discard chromosomes whose fitness values fall below the predefined threshold.
6. While the maximum number of generations has not been reached do:

project evaluation, an independent dataset from a similar project was used. The Random Forest classifier available in WEKA 3.8 was configured with 100 trees due to its high classification performance. The remaining parameters were retained at their default settings, including bag size = 100%, minimum instance weight = 1.0, minimum variance for split = 0.001, and random seed = 1. The number of attributes considered at each split (K) was automatically determined by WEKA. A forest size of 100 trees was selected to provide a balance between predictive performance and computational efficiency. The combination of cross-validation and cross-project testing ensured robust evaluation.

3.5 Computational Complexity and Runtime Environment

The computational cost of the proposed approach is primarily associated with the Genetic Algorithm (GA)-based feature optimization process and the Random Forest classification model. Let P denote the population size, G the number of generations, F the feature dimension, and K the number of cross-validation folds. The approximate computational complexity of the GA component can be expressed as: O (P×G×F×K). This complexity arises because the fitness of each chromosome is evaluated repeatedly across generations using stratified 10-fold cross-validation.

The computational complexity of the Random Forest classifier depends on the number of trees, the number of training instances, and the feature dimensions. By reducing the feature space to the top 10 ranked features, the proposed method decreases the search space, thereby improving computational efficiency during the optimization process and reducing classification overhead. The experiments were executed on a Dell workstation equipped with an Intel Core i7 processor and 16 GB RAM.

3.6 Performance Evaluation Metrics

Model performance was evaluated using precision, recall, and F-measure. Recall measures the ability to detect actual vulnerabilities, precision reflects prediction accuracy, and F-measure provides a balanced assessment.

Table 2. Performance Evaluation Summary: Within-project Predictions

Dataset	Prediction Type	Recall	Precision	F-measure
Tomcat 9.0.0.M1	Within	95%	94%	95%

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

Dataset	Prediction Type	Recall	Precision	F-measure
Jetty 9.4.RC1	Within	98%	95%	95%

4.1 Result of the Within-project Predictions

The results of the within-project predictions are summarized in Table 2, followed by a brief discussion.

Discussion

As shown in Table 2, the proposed model achieved strong within-project prediction performance on both Tomcat 9.0.0.M1 and Jetty 9.4.0.RC1. The high recall values indicate that most vulnerable files were successfully identified, thereby reducing the likelihood of missed vulnerabilities (false negatives). Similarly, the high precision values suggest that relatively few non-vulnerable files were incorrectly classified as vulnerable, reducing false alarms (false positives). These results indicate that the selected feature subset effectively captures vulnerability-related characteristics within individual software projects. Furthermore, the consistently high F-measure values demonstrate that the proposed approach maintains a strong balance between vulnerability detection and prediction accuracy. Overall, the findings demonstrate the effectiveness of the proposed model, which combines modified GA-based feature optimization with Random Forest classification.

4.2 Result of the Cross-project Predictions

Cross-project prediction results are summarized in Table 3 and further illustrated in Figure 3, followed by a brief discussion in the next subsection. As shown in Table 3 and Figure 3, the cross-project performance of Tomcat 9.0.0.M1 on Jetty 9.4.0.RC1 achieved a recall of 81.9%, a precision of 79.1%, and an F-measure of 76.9%. The relatively high recall indicates that most vulnerable files in the target project were correctly identified, thereby limiting false negatives. Although precision is lower than the corresponding within-project result, the model still maintains a reasonable level of false-positive control. These findings suggest that vulnerability-related patterns learned from Tomcat remain applicable to Jetty and that the optimized feature subset captures characteristics that are transferable across projects.

Table 3. Performance Evaluation Summary: Cross-project Predictions

Train → Test	Prediction Type	Recall	Precision	F-measure
Tomcat → Jetty	Cross-project	81.9%	79.1%	76.9%
Jetty → Tomcat	Cross-project	75.5%	70.2%	70.9%

Discussion

For Jetty 9.4.0.RC1 evaluated on Tomcat 9.0.0.M1, the recall, precision, and F-measure are 75.5%, 70.2%, and 70.9%, respectively. The lower recall suggests an increase in false negatives, while the reduced precision indicates a higher number of false-positive predictions. Despite this decline, the model retains useful predictive capability across projects, indicating a degree of feature transferability and robustness.

The reduction in performance compared with the Tomcat → Jetty scenario suggests that knowledge transfer from Jetty to Tomcat is more challenging, possibly due to differences in project characteristics, coding practices, and vulnerability distributions.

The reduction in cross-project performance compared with within-project prediction can be attributed to differences in project characteristics, coding practices, and vulnerability distributions between Tomcat and Jetty. Such differences may alter the relationships between software metrics and vulnerability labels, making knowledge transfer more challenging. Nevertheless, the model maintained reasonable predictive performance across projects, indicating that the optimized feature subset captures vulnerability-related characteristics that are transferable beyond the training project.

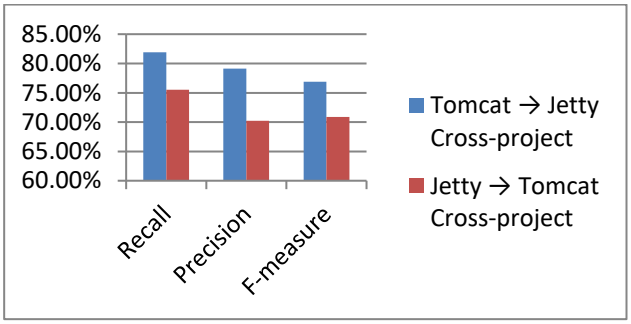

Fig. 3. Cross-project Prediction Performance of Tomcat 9.0.0.M1 and Jetty 9.4.0.RC1

4.3 Comparative Analysis

Table 4 presents a comparative summary of recent studies on software vulnerability prediction models (VPMs), highlighting the datasets used, feature engineering strategies, classification models, and reported performance metrics. The comparison provides a basis for evaluating current

4 RESULTS AND DISCUSSION

This section presents model validation results, beginning with within-project prediction performance.

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

research trends and positioning the proposed approach relative to recent studies in the literature. The reviewed studies employ techniques such as Random Forest (RF), Genetic Algorithm (GA), Bidirectional Long Short-Term Memory (BiLSTM), Att-BiLSTM = Attention-based Bidirectional Long Short-Term Memory, Quantum Convolutional Neural Network (QCNN), and Trace Gadgets (TG).

Discussion

Deep learning methods such as QCNN and Att-BiLSTM (Hussain et al., 2024; Zhao et al., 2024) report high accuracies of 99.2% and 95% respectively, but often focus mainly on accuracy, limiting

Table 4. Comparative Analysis of Recent Software Vulnerability Prediction Studies

Authors	Year	Dataset	Method / Feature Strategy	Model	Performance (Precision / Recall/ F-measure / Accuracy)	Notes
Hussain et al.	2024	Java datasets	QCNN-based feature learning	QCNN	— / — / — / 99.2	High accuracy
Hussain et al.	2024	Java systems	Improved QCNN features	QCNN	— / — / — / 99.0	Variant model
Zhao et al.	2024	Java systems	Attention-based sequence features	BiLSTM	— / — / — / 95.0	Accuracy only
Hussian, Ghafoor, Perviaz and Fakhar	2025	SARD (Java)	BiLSTM feature learning	BiLSTM	88.6 / 87.9 / 88.2 / 89.1	Full metrics
Nguyen et al.	2025	FFmpeg, Linux	Vulnerability-aware features	VulGuard	— / — / 90.9 / 80.9	JIT prediction
Mächtle et al.	2025	JVM systems	Trace-based feature reduction	GPT-4o + TG	— / — / 77–81 / 73	Context study
This study	**2026**	**Jetty**	**GA-based feature selection**	**RF**	**98 / 95 / 95 / —**	**Within-project**
This study	**2026**	**Cross-project Tomcat → Jetty**	**GA transferable features**	**RF**	**79.1 / 81.9 / 76.9 / —**	**Cross-project**

comprehensive comparison using precision, recall, and F1-score. Although several recent studies report high accuracy values, accuracy alone is insufficient for evaluating vulnerability prediction models due to the inherent class imbalance in software datasets . Metrics such as precision, recall, and F-measure provide a more reliable assessment of model performance, particularly in security-critical applications where missed vulnerabilities can have severe consequences. Traditional ML approaches also lack complete evaluation reporting, making performance interpretation difficult. In contrast, BiLSTM-based models (Hussain et al., 2025) provide more balanced results with 88.6% precision, 87.9% recall, 88.2% F1-score, and 89.1% accuracy, indicating more stable learning behavior. However, Nguyen et al. (2025) and Mächtle et al. (2025) still show partial metric reporting and performance variability, especially in hybrid and LLM-based approaches.

A consistent limitation across prior work is weak cross-project generalization. While within-project performance is strong, models degrade significantly under dataset shift conditions. Mächtle et al. (2025), for example, shows performance dropping from 78.8% to 66.7% in different settings, highlighting instability across environments. In comparison, the proposed model achieves strong within-project performance, with precision ranging from 94% to 95%, recall ranging from 95% to 98%, and F-measure consistently at 95% across the Tomcat and Jetty datasets. Although cross-project performance decreases, with F-measure scores of 76.9% and 70.9%, the model maintains a reasonable level of predictive capability when applied to independent software projects. These results suggest that the selected feature subset retains useful predictive information across projects, indicating a degree of feature transferability and robustness despite differences in project characteristics and vulnerability distributions.

4.4 Threats to Validity

Construct validity is addressed by combining traditional software metrics with vulnerability-

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

http://dx.doi.org/10.46792/fuoyejet.v11i3.1565 engineering.fuoye.edu.ng/journal

specific indicators extracted from Java source code. Genetic algorithm-based feature selection further improves reliability by removing irrelevant features and reducing noise. Internal validity is strengthened through a controlled experimental design using RF classification, consistent preprocessing, and stratified cross-validation. These steps ensure performance differences are due to the model design rather than experimental bias. External validity is limited as the study focuses on Apache Tomcat 9.0.0.M1 and Eclipse Jetty 9.4.0.RC1. While these are industrial systems, results may not fully generalize to other Java applications or software ecosystems. Cross-project evaluation partially mitigates this limitation but broader validation is still required. Scalability is another limitation, as the datasets used are smaller than large-scale industrial repositories. Performance under larger datasets or real-time deployment scenarios remains to be explored.

5 CONCLUSION

This study presented a Java-based vulnerability prediction model (VPM) that integrates a modified Genetic Algorithm (GA)-based feature optimization approach with a Random Forest (RF) classifier. Evaluated on the Apache Tomcat and Eclipse Jetty datasets, the proposed model achieved strong within-project performance, with recall, precision, and F-measure values ranging from 94% to 98%. The results demonstrate that combining traditional software metrics with security-specific indicators improves vulnerability prediction by capturing security-relevant characteristics more effectively. A major contribution of this study is the introduction of the proposed Extreme Two Genes (ETG) crossover operator, which modifies the crossover stage of the Genetic Algorithm to improve feature optimization. The optimized feature subsets enhanced classifier performance by reducing irrelevant attributes while retaining a degree of transferability across software projects. Although cross-project prediction performance declined because of differences in project characteristics and data distributions, the results indicate that the proposed approach maintains reasonable predictive effectiveness under dataset-shift conditions.

The proposed model has practical relevance for industrial software development environments. It can be integrated into software testing and code review processes to support early vulnerability detection by prioritizing files that are more likely to contain vulnerabilities. Rather than replacing static analysis tools, the model complements them by providing an additional predictive layer that helps developers and quality assurance teams focus security inspection efforts more effectively, potentially reducing both false alarms and missed vulnerabilities. Future work will focus on improving cross-project generalization through domain adaptation, transfer learning, and other knowledge-transfer techniques. Additional research will also investigate larger and more diverse vulnerability datasets, as well as hybrid learning approaches that combine traditional machine learning with deep learning methods to further enhance prediction performance and robustness.

REFERENCES

Bajantri, G., & Shariff, N. C. (2025). *Enhancing software security with CNN-LSTM hybrid model: Vulnerability detection* [Preprint]. Research Square. https://doi.org/10.21203/rs.3.rs-8151185/v1

Bassi, D., & Singh, H. (2024). *A comparative analysis of metaheuristic feature selection methods in software vulnerability prediction*. e-Informatica Software Engineering Journal, 19(1), 250103.

Dam, H. K., Tran, T., Pham, T., Ng, S. W., Grundy, J., & Ghose, A. (2017). *Automatic feature learning for vulnerability prediction* (arXiv Preprint No. arXiv:1708.02368). arXiv. https://arxiv.org/abs/1708.02368

Gunda, S. K. (2025, July). Automatic Software Vulnerabilty Detection Using Code Metrics and Feature Extraction. In *2025 2nd International Conference On Multidisciplinary Research and Innovations in Engineering (MRIE)* (pp. 115-120). IEEE.

Hussian, S., Ghafoor, M. I., Perviaz, P., & Fakhar, S. (2025). *Code-semantic based intelligent vulnerabilities detector*. Spectrum of Engineering Sciences, 117–130.

Hussain, S., Nadeem, M., Baber, J., Hamdi, M., Rajab, A., Al Reshan, M. S., & Shaikh, A. (2024). *Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction*. Scientific Reports, 14(1), 7406.

Khare, A., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., & Naik, M. (2025, March). *Understanding the effectiveness of large language models in detecting security vulnerabilities*. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)* (pp. 103–114). IEEE.

Le, T. H. M., & Babar, M. A. (2024, October). *Mitigating data imbalance for software vulnerability assessment: Does data augmentation help?* In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 119–130).

Manjula, C., & Florence, L. (2018). *Hybrid approach for software defect prediction using machine learning with optimization technique*. International Journal of Computer and Information Engineering, 12(1), 28–32.

Mächtle, F., Loose, N., Schulz, T., Sieck, F., Serr, J. N., Möller, R., & Eisenbarth, T. (2025). *Trace gadgets: Minimizing code context for machine learning-based vulnerability prediction*. arXiv preprint arXiv:2504.13676.

Nguyen, D., Tran-Duc, M., Le-Cong, T., Le, T. H. M., Babar, M. A., & Huynh, Q. T. (2025). *VulGuard: An unified tool for evaluating just-in-time vulnerability prediction models*. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 945–949). IEEE.

Osman, H., Ghafari, M., Nierstrasz, O., & Lungu, M. (2017). *An extensive analysis of efficient bug prediction configurations*. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2017)*.

Scandariato, R., Walden, J., Hovsepyan, A., & Joosen, W. (2014). *Predicting vulnerable software components via text mining*. IEEE Transactions on Software Engineering, 40(11), 993–1006.

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)

Shamal, P. K., Rahamathulla, K., & Akbar, A. (2017, March). A study on software vulnerability prediction model. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)* (pp. 703-706). IEEE.

Sultana, K. Z., Anu, V., & Chong, T. Y. (2024). Using software metrics for predicting vulnerable classes in java and python based systems. *Information Security Journal: A Global Perspective*, *33*(3), 251-267.

Thomas, N. S., & Kaliraj, S. (2024). *An improved and optimized random forest based approach to predict the software faults*. SN Computer Science, 5(5), 530.

Tran, N. K., & Cheng, W. (2018). Multiplicative tree-structured long short-term memory networks for semantic representations. In Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics (pp. 276-286).

Wang, H., Tang, Z., Tan, S. H., Wang, J., Liu, Y., Fang, H., Xia, C., & Zheng, W. (2024). *Combining structured static code information and dynamic symbolic traces for software vulnerability prediction*. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)* (pp. 169:1–169:13).

Zhao, S., Zhu, J., & Peng, J. (2024). *Software vulnerability mining and analysis based on deep learning*. Computers, Materials & Continua, 80(2), 3263.

© 2026 The Author(s). Published by Faculty of Engineering, Federal University Oye-Ekiti.
This is an open access article under the CC BY NC license. (https://creativecommons.org/licenses/by-nc/4.0/)