University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)

ISSN: 2714-3627

A Journal of the Faculty of Computing, University of Ibadan, Ibadan, Nigeria

Volume 16 No. 1, January 2026

journals.ui.edu.ng/uijslictr
http://uijslictr.org.ng/
uijslictr@gmail.com

University of Ibadan
Journal of Science and Logics in ICT Research

Software Security Vulnerability Prediction Modelling for C/C++ Systems

[1]✉ Olatunji M. A., [2]Ujah-Ogbuagu C. B., [3]Adebayo P. O., and [4]Agbolade S. J.

Redeemer's University, Ede, Osun State, Nigeria
National Defence College Nigeria, Abuja, Nigeria
University of Ilorin, Ilorin, Nigeria
Redeemer's University, Ede, Osun State, Nigeria

olatunjim@run.edu.ng, bcujah-ogbuagu@ndc.gov.ng, adebayo.po@unilorin.edu.ng, agbolades@run.edu.ng

Abstract
This study focused on developing realistic software security Vulnerability Prediction Models (VPMs) for C/C++ systems. The aim is to mitigate security vulnerabilities and prevent exploitation in C/C++ projects by identifying vulnerable source files for patching before deployment. The study addressed the limitations of existing software VPMs, such as low accuracy, poor traceability of vulnerabilities, dataset imbalance, and the use of irrelevant metrics. The research used relevant security-related metrics as features and addressed the dataset imbalance issue by oversampling. Genetic algorithm was modified to overcome local optima problem and in turn used to optimize the correlation values of the metrics and improved the performance of random forest classifier. The study also highlighted that oversampling improved predictability and feature elimination mitigated overfiting. The developed software VPMs exhibited high performance in cross-project predictions, with recall, precision, and f-measure exceeding 80%, surpassing most performance reported in the literature. The software VPMs enable easy traceability of vulnerable components. Therefore, the study recommended the adoption of these software VPMs by quality assurance teams in software development companies to predict vulnerable files to patch before deployment. Additionally, the primary dataset used in the study is recommended as a benchmark for software VPMs researchers.

***Keywords*: Vulnerability, Metrics, Cross-project, Correlation, Model, Optimization**

1. Introduction

C and C++ programming languages are widely used for developing various software systems, including operating systems, applications, and databases. However, these languages are prone to security vulnerabilities due to memory-related flaws, common programming errors, and failure to follow security guidelines during development. Buffer overflow vulnerabilities, caused by writing outside allocated memory, are common issues in C/C++ due to the lack of built-in boundary check features [1]. These vulnerabilities can lead to program crashes, memory corruption, and even the execution of malicious code, compromising the confidentiality, integrity, and availability of the system [2], [3]. Other vulnerabilities include memory leaks, cross-stack access, mismatched allocation, invalidated input, injection attacks, incorrect type conversion, directory traversal, input validation, and cross-site scripting. Such vulnerabilities are frequently reported in the Common Vulnerability and Exposures (CVE) database.

Exploiting software vulnerabilities can result in data loss, monetary loss, manipulation, downtime, data breaches, and reputational damage [4]. For example, the Heart bleed vulnerability in OpenSSL affected millions of internet users in 2014. Trust wave cybersecurity experts also reported security vulnerabilities in Microsoft Windows Object Linking and Embedding (OLE), which were exploited in Russian cyber-espionage campaigns against NATO, energy sector firms, and government organizations.

To address vulnerability exploitation, the discovery of vulnerable components in software systems is crucial. Automatic Static

Olatunji M. A., Ujah-Ogbuagu C. B., Adebayo P. O., and Agbolade S. J. (2026). Software Security Vulnerability Prediction Modelling for C/C++ Systems, *University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)*, Vol. 16 No. 1, pp. 15 - 26

Analysis (ASA) tools are commonly used to detect security-related flaws early in the development process by analyzing the source code. However, ASA tools produce false positives due to predefined rules used for vulnerability detection. To improve vulnerability detection, a software security Vulnerability Prediction Model (VPM) is needed in addition to ASA tools [5]. Software VPMs leveraging on machine learning algorithms trained on labeled data that include security-related metrics and traditional software metrics can predict vulnerabilities in test data with lower false positives [6]. Thus aids the production of secure software.

Existing software VPMs have shown limited accuracy in cross-project prediction, and most of them rely on statistical methods and machine learning techniques [7], [8], [9]. This paper proposes a search-based approach for feature selection and training; specifically, using a modified Genetic Algorithm (GA) to train a random forest classifier for predicting vulnerabilities in C/C++ cross-projects.

2. Related Works

Research in software security vulnerability prediction for C/C++ systems has explored various approaches and techniques. Manjula and Florence [10] improved the optimization performance of GA by modifying the chromosome and fitness function. The study achieved over 80% accuracy in predicting software defects with the GA-Decision Tree approach. Zimmermann, Nagappan, and Williams [11] used Support Vector Machine (SVM) with traditional metrics but obtained relatively low precision and recall values, indicating inconsistent and low performance in vulnerability prediction for Windows Vista (C/C++ system).

Morrison, Herzig, Murphy, and Williams [12] found that software VPMs constructed at the binary level were accurate but too large for vulnerable components traceability, while models at the module/class level granularity were less accurate. They recommended incorporating security domain knowledge to improve VPM performance.

Alves, Fonseca, and Antunes [13] used software metrics to predict security vulnerabilities in C/C++ open source projects. Random Forest (RF) achieved the best results in the study, but overall machine learning techniques fell short of expectations for actionable models. Jimenez [14] analyzed vulnerabilities in the Linux kernel and OpenSSL. They identified 9 prevalent and 5 critical vulnerability types. They recommended future research on models targeting the critical types and emphasized the need for project-specific approaches. Shin, Meneely, Williams, and Osborne [15] evaluated complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. Metrics values from C/C++ systems were used as features for software VPM but achieved low accuracies of around 70%.

Filus, Boryszko, Domanska, Siavvas, and Gelenbe [16] explored the evaluation and selection of vulnerability features derived from static analysis of C/C++ source files; identifying size, complexity, code smells, and technical debts as good indicators. They highlighted the importance of recall and specificity as performance measures for software VPMs. Zhang, Wang, Li, Sun, Zhang, Ma and Liu [17] explored the capabilities of Large Language Models (LLMs) including ChatGPT for automated vulnerability localization in C/C++ systems; finding fine-tuning effective with sufficient training data but struggles with limited data. While generalizability is good, subtle and novel vulnerabilities require careful treatment. However, the performance often below 80% recall and precision in almost all cases.

Kile, Garba, and Bassi [18] developed *Identi-fix*, an automated Support Vector Machine-based software vulnerability scanner trained on 100 preprocessed C/C++ functions with 527 engineered features. Evaluated with stratified train-test splits and five-fold cross-validation, it achieved 86.2% accuracy, 84.8% precision, 88.0% recall, and 86.4% F1-score. The study demonstrated effective early vulnerability detection and practical integration into software workflows but focused solely on within-project prediction, while cross project prediction was not addressed.

Ahmed, Harzevili, Shin, Pham, and Wang [19] introduced *SECVULEVAL*, a statement-level benchmark for LLM vulnerability detection in 25,440 C/C++ functions spanning 5,867 CVEs. Evaluating models like GPT-4.1 and Claude-3.7-Sonnet revealed poor statement-level detection (best F1-score 23.83%, precision 15.35%) and difficulties in capturing context such as type definitions, global variables, and interprocedural dependencies. The study highlighted the limitations of current LLMs for fine-grained, context-rich vulnerability detection. In the same vein; Huynh, Zhang, Jayasundera, Jeon, Kim, Bi, and Hong [20] proposed an LLM framework using *prompt engineering* for C/C++ systems vulnerability detection and patching on DiverseVul. GPT-4o achieved the highest correction rate (45.77%) versus 21.56% for GPT-3.5. Strengths included automated detection and patching capabilities, while weaknesses involved poor CWE classification (F1 0.16) and challenges in validating automatic code fixes.

This research developed a practical software VPMs for C/C++ systems. It utilized traditional software metrics and security-related metrics as features, resolved multi-collinearity and traceability issues, and modified GA feature selector to overcome local optima issue peculiar to GA. Modified GA selected and optimized features while RF of optimized hyper-parameters was employed for classification.

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 C/C++ systems.

3.1 Identification and Collection of Vulnerable C/C++ Project Samples: Dataset Projects

The software projects considered in this modeling research as datasets were identified and selected based on the following criteria.

1. The C/C++ projects are open source to avoid licensing restrictions.
2. C/C++ projects with high number of reported vulnerabilities on CVE. Vulnerable files in the plug-ins and third-party components are also considered. Using these criteria, two C/C++ projects presented in Table 1 were identified. The source code files of the projects were collected from GitHub.com.

3.2 Feature Creation

The predictor variables that constitute the feature instances in this research were originally created and labeled, and is void of duplication that is often found in large dataset [21]. This study leverages on relevant feature creation and optimization to achieve its aim of high recall and precision in software security VPM performance. Table 2 presents the tools that were used to measure the metrics that constitute the features.

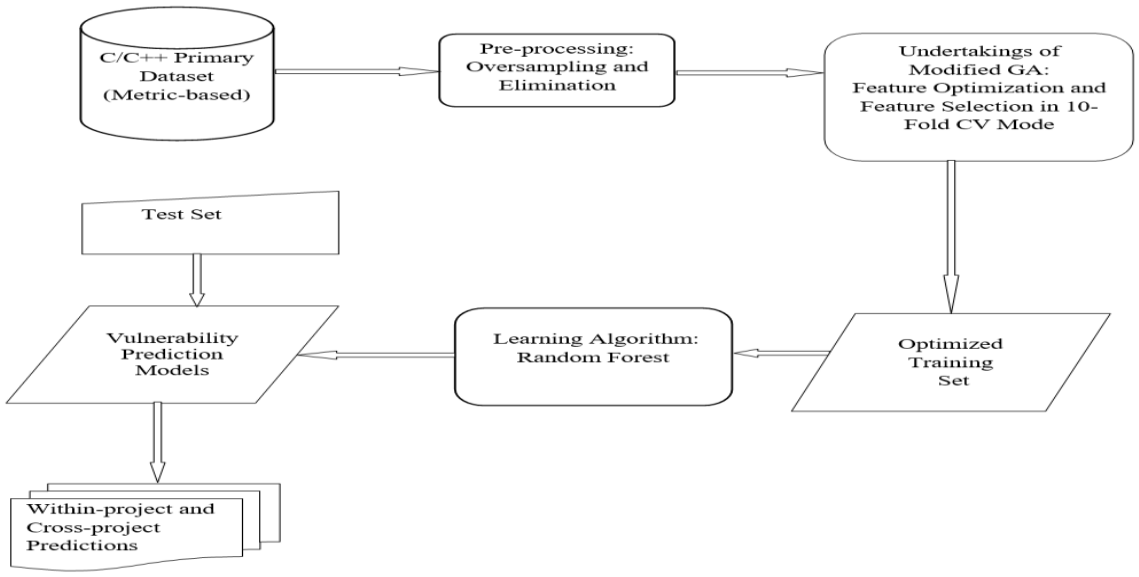

Figure 1: Framework for the Study

Table 1: Description of Selected Software Projects

Software Project	Description
My Structured Query Language (MySQL) 5.1.30 (C/C++)	MySQL is an open-source relational database management system. It uses SQL (Structured Query Language) to manage data inside the database. 310 C/C++ source files were extracted in MySQL 5.1.30
Apache HTTP Server 2.4.9 (C/C++)	Apache is an open source server application that accepts directory requests from Internet users and sends corresponding information like files or Web pages. 310 C/C++ source code files were extracted in Apache HTTP Server 2.4.9

Table 2: Measurement Tools

S/N	Measurement Tool	Description	Metrics Measured
1.	Yasca static analyzer	Yasca utilizes nine static analyzers like PMD, FindBugs, and Cppcheck to scan code for vulnerabilities and conformance to best practices across various languages, while also employing custom scanners for additional analysis.	SQL Injection, Cross-Site Scripting, Bad Practice, Denial of Service, Poor Logging Practice, Authentication issues, Code Quality Issues, Cryptography, Use After Free, Function, Sensitive Improper Null Termination
2.	Code Line Counter (CLC)	CLC counts Lines of Code (LOC), number of blank lines, and number of mixed lines (code and comments) amongst others. It was used to compute churn (code changes) in this research.	LOC, Churn
3.	Resource Standard Metrics (RSM) analyzer	RSM analyzes C/C++ and Java files, providing reports on metrics like LOC, number of modules, and complexity, with transparent computation methods, making it reliable for code analysis, including Code Churn and complexity metrics.	Cyclomatic Complexity, Interface Complexity, Comment Lines, Lines of Code, Churn.

Waikato Environment for Knowledge Analysis (WEKA): versatile toolkit/platform for data mining, featuring machine learning and statistical algorithms, plus tools for preprocessing, classification, regression, clustering, association rules mining, and visualization.

The files in the C/C++ software systems were labeled by CVE their details, security advisories, information on the developers' websites and manual review. CVE is a well-accepted catalog of common coding and design issues that may affect software security. The dataset files were labeled as vulnerable or non-vulnerable.

3.3 Data Pre-processing: Resampling and Feature Elimination

To address dataset imbalance, random oversampling with replacement was applied using WEKA's Resample class. Feature elimination was performed by considering only the top 10 most correlated features using the CorrelationAttributeEval and Ranker in WEKA's preprocessing section.

3.4 Modification of GA, Feature Selection and Optimization

The GA for feature selection in this study was modified to include Extreme Two Genes (ETG) crossover operator. The GA uses binary encoding to represent feature subsets, with each chromosome being a string of binary 1s and 0s. The fitness function evaluates the subsets' worth using the Correlation-based Feature Subset Evaluator (CfsSubsetEval), considering both predictive ability and redundancy. The selection phase chooses parents based on correlation merit, and the ETG crossover operator exchanges bit segments at the extreme ends of the cut points to create diverse offspring. The implemented modified GA was integrated into the WEKA simulation platform. The modified GA with ETG crossover operator enhances diversity and prevents premature convergence; thus improves feature selection and optimization process. The optimal training subsets, selected based on their performance and correlation with the class attribute, are used for a random forest classifier. The algorithm of the modified GA is presented in Algorithm 1.

Algorithm 1. Modified Genetic Algorithm

_ Input: Training Set {ARFF of MySQL, Apache HTTP Server}
Output: Optimized Training Set, Selected Features

Step 1: Setting Parameters
1.1 Input crossover probability, pCrossover
1.2 Input maximum generation, maxGenerations
1.3 Input mutation probability, pMutation
1.4 Input population size, maxPopulation
1.5 Input report frequency, m_reportFrequency

Step 2: Generate Initial Population of Individuals
2.1 Genes = features in training set
2.2 Chromosome (individual) = file level instance features
2.3 population = ∀ chromosome ϵ training set, of even number
2.4 Binary encoding: convert all genes to binary equivalent using parseInt()

Step 3: Compute Fitness Function
3.1 Compute correlation coefficient (r) for ∀ chromosome in relation to their Vulnerability status using

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

Where x rep chromosome, and y rep vulnerability status

3.2 feature or chromosome frequency in 10-fold CV: ∝ (**r** + collinearity)
3.3 set **r** threshold for the generation.

Step 4: Selection Operation
4.1 Set chromosome (n) = maxPopulation number
4.2 For (i=1, chromosome (i) ≤ chromosome (n), i++)
4.3 If chromosome (i) **r** value is < threshold **r**
4.4 Then discard the chromosome (i)
4.5 Else enlist the chromosome (i) to next generation
4.6 End if
4.7 End for

Step 5: Modified Crossover Operator: Extreme Two Genes (ETG) Crossover

5.1 number of crossing chromosomes = c // based on crossover probability
5.2 Chromosome (i) = {gene1, gene2, gene3, gene4, … ,gene9, gene n}
5.3 Chromosome (i +1) = {gene1, gene2, gene3, gene4, … , gene9, gene n}
5.4 Set newChromosome (i) = { }
5.5 Set newChromosome (i+1) = { }
5.6 Population = {chromosome1, chromosome2, chromosome3, …, chromosome n}
5.7 If maxPopulation Number ≠ even number,
5.8 Then Output "maxPopulation must be even number"
5.9 Else
5.10 For (i =1, i ≤ c, i++)
5.11 If chromosome (i) genes > 4
5.12 Do pair the chromosome (i) and chromosome (i +1)
5.13 ∀ paired chromosomes ϵ c, set cut point1 = at gene2
5.14 Set cut point2 = gene (n – 2)
5.15 maxNumber of pair = c/2,
5.16 For (i=1, i ≤ c/2, i++)
5.17 Swap: gene1, gene2, gene (n -1), gene n
5.18 Swapped chromosomes = new chromosome (i) &
5.19 new chromosome (i+1)
5.20 End for
5.21 End if
5.22 End for
5.23 End if

Step 6: Mutation Operation

6.1 mutable Chromosomes = m // the limitation in number is by mutation probability
6.2 For (i = 1, I ≤ m, i++) Do
6.3 If Chromosome (i) ≡ any of chromosome m
6.4 Then mutate a binary bit of a gene in repeated Chromosomes
6.5 Else if Chromosome (i) ≠ any of Chromosome m
6.6 Then do no mutation
6.7 End if
6.8 End if
6.9 End for

Step 7: Stopping Criteria

7.1 If generation < 100th
7.2 Go to step 3
7.3 Else if generation= 100th
7.4 Do: Evaluate and replace less fit individuals
7.5 Output: optimized training set (1st& 100th generation)
7.6 Output: 10-fold CV feature selection
7.7 End if
7.8 End if
7.9 End

Table 3: Performance Evaluation Summary: Within-project Predictions

PHP Datasets	Prediction Type	Recall	Precision	F-Measure
Apache HTTP Server 2.4.9	Within-project	93%	91.4%	89.4%
SQL 5.1.30	Within-project	91.9%	91.8%	91.5%

3.5 Data Splitting, Supplied Test Set and Classification

The optimized training set was split into two subsets: one for training the model for within-project prediction and the other for evaluation/testing using Pareto principle of 80:20 [22]. For cross-project prediction, a test set from a similar project was used for evaluation. Random forest was chosen as the classifier because of its high performance [23], [24], [25]. It combines multiple decision trees to make predictions based on majority voting, with 100 decision trees used to prevent overfitting and improve accuracy.

3.6 Performance Evaluation Metrics

The performance of the prediction models in this study was evaluated using recall, precision and f-measure. Recall measures the ability to detect true positives and prevent false negatives. Precision quantifies the proportion of correct positive predictions, reducing false positives. The F-measure combines precision and recall into a single score, providing a balanced

evaluation metric. Precision, recall, and F-measure were chosen as performance measures due to their consideration of the costs associated with false negatives and false positives, as well as the objective of minimizing wasted time on non-vulnerable files.

4. Result and Discussion

This section presents the validation of the software VPMs on the test set to predict the vulnerable files in C/C++ projects. The results of the within-project predictions were presented and discussed first, followed by cross-project predictions.

4.1 Result of the Within-project Predictions

The following are the model's pertinent information: Instances in MySQL: 310, Instances in Apache HTTP Server 2.4.9: 310, Test mode: split 80.0% train, 20% remainder test. Classifier: random forest of 100 decision trees

4.2 Discussion

Table 3 shows that within-project prediction for SQL 5.1.30 achieved high performance measures: recall of 90.3%, precision of 91.4%, and f-measure of 89.4%. These values indicate a low chance of vulnerabilities slipping through, with less than 10% false negatives and very few false positives. Similarly, within-project prediction for Apache HTTP Server 2.4.9 achieved recall, precision, and f-measure values all above 90%. These results demonstrate reliable and actionable predictions with low false positives and false negatives.

4.3 Result of Cross-project Predictions

Software VPM trained with the instances of MySQL 5.1.30 was validated on test set from Apache HTTP Server 2.4.9 and vice-versa. Attributes Optimizer and Selector: modified GA, Classifier: random forest (having 100 decision trees).

4.4 Discussion

As stated in Table 4, cross-project performance of SQL 5.1.30 on Apache Server 2.4.9 test set: recall is 87.1%, precision is 88.8% and f-measure is 83.3%. This cross-project prediction is good to guide the review team to the vulnerable files. To locate the vulnerable files after the prediction results is out, is by relating the vulnerable instance lines on the predicted result output to the corresponding source code file table of the supplied test set. Performance less than 80% is not acceptable in software engineering [23], [26]. The prediction is of both low false positives and low false negatives; is practicable and reliable.

Cross-project performance of Apache Server 2.4.9 on MySQL 5.1.30 test set: recall is 81.9%, precision is 80.6% and f-measure is 81.1%. This cross-project performance is good, is a sure guide for the testers in locating the vulnerable source files. The three evaluation measures are all over 80%, this means there is low false negatives and low false positives in the prediction. Vulnerabilities would scarcely escape to deployment. Similarly, precious time of the testers and quality assurance resources would not be wasted on much false positives.

4.5 Comparative Analysis

Firstly, the performance of the modified GA optimizer was evaluated by comparing the prediction performance of random forest trained by different search-based techniques.
Secondly, the performance results of random forest trained by the modified GA were compared with reported C/C++ software security vulnerability prediction results in the recent literature.

Table 4: Performance Evaluation Summary: Cross-project Predictions

C/C++ Datasets	**Prediction Type**	**Recall**	**Precision**	**F-Measure**
Training set: Apache HTTP Server 2.4.9 Test set: MySQL 5.1.30	Cross-project	87.1%	88.8%	83.3%
Training set: MySQL 5.1.30 Test set: Apache HTTP Server	Cross-project	81.9%	86%	81.1%

4.5.1 Performance Evaluation of Modified GA

The Random Forest consists of 100 Decision Trees. The parameters of the modified GA are: crossover = 0.8, the mutation probability = 0.005, maximum generations = 100, population size = 310 and report frequency = 100. The datasets used for model building is Apache HTTP Server 2.4.9, and it was validated on MySQL 5.1.30 for cross project prediction. The attribute selection mode is 10 cross-validation. Other search-based feature optimizers were also used to train random forest.

4.5.1.1 Discussion

Here the performance metrics obtained from random forest classifier, trained with modified GA were slightly higher than the results from other search-based feature optimizers as seen in Table 5. It is noteworthy that all the feature optimization techniques performed well in the model building, they all hit the acceptable threshold of 80% for performance measures. The difference in performance measures are close, search-based algorithms are seen to be good for feature optimization.

4.5.2. Evaluation of Software VPMs Performance by Comparison with Recent Works

The performance of software VPMs in recent studies were compared with the performance measures in this study. Within-project and cross-project prediction from the main approaches to software VPMs were considered. Table 6 shows the performance of diverse software VPMs as reported in the literature. In cases where two or more models were built in a work, the performance of the best performing model was enlisted in the Table.

Table 5: Random Forest Trained by Search-based Techniques for Cross-project Prediction

Cross-project Prediction Performance Measures by Random Forest. Training set: Apache HTTP Server 2.4.9, Test set: MySQL 5.1.30 (C/C++ projects)			
Search-based Feature Selection Techniques	**Precision**	**Recall**	**F-Measure**
Modified GA	**88.8%**	**87.1%**	**83.3%**
GA	85.3%	84.5%	81.2%
Evolutionary Programming	85.3%	84.5%	81.2%
Exhaustive Search	85.3%	84.5%	81.2%
Greedy Stepwise	86.5%	84.7%	81.3%
Random Search	85.3%	84.5%	81.2%
Tabu Search	85.3%	84.5%	81.2%
Scatter Search V1	85.3%	84.5%	81.2%
Best First Search	85.3%	84.5%	81.2%

Table 6. Within-project and Cross-project Prediction Performance Comparison

Authors	**Dataset Description**	**Feature Selection or contribution**	**Classifiers**	**Precision, Recall, F-Measure**
This study	**Security related and traditional software metrics from C/C++ projects**	**Modified GA**	**RF (within-project)**	**91.4% 93% 89.4%**
Medeiros, Ivaki , Costa and Vieira [27]	Traditional software metric-based from C/C++ Systems	Dimensional reduction: Correlation and redundancy analysis.	Random Forest, Extreme Boosting (within-	(Recall) 90% (F-measure) 34

		Application scenario criticality definition	project)	
Hin, Kan, Chen and Babar [28]	Graph-based Features and Text mining features from C/C++ Systems	LineVD framework leveraging on control and data dependencies between statements	Graph Neural Networks (within-project)	27% 53% 36%
Kalouptsoglou, Siavvas, Kehagias, Chatzigeorgiou and Ampatzoglou [29]	Traditional software metric-based features. Language agnostic approach	Point-BiSerial Correlation	RF SVM (within-project)	90% 68% 77% 94% 57% 71%
Kile, Garba, and Bassi [18]	100 preprocessed C/C++ code functions	Structural, lexical, and vulnerability-related features (527-dimensional)	SVM	84.8% 88% 86.4
Ahmed, Harzevili, Shin, Pham, and Wang [19]	SECVULEVAL: 25,440 C/C++ functions, statement-level annotations for 5,867 CVEs	Fine-grained contextual annotations for statement-level vulnerability detection	LLMs (GPT-4.1, Claude-3.7-Sonnet, etc.)	15.35%(Precision) 23.83% (F-Measue)
Huynh, Zhang, Jayasundera, Jeon, Kim, Bi, and Hong [20]	DiverseVul dataset of C/C++ vulnerabilities; Purple Llama CyberSecEval for patching	Prompt engineering: role-based, zero-shot chain-of-thought, structured prompts	LLMs (GPT-3.5, GPT-4o, Gemini 2.0 Flash, Meta Llama 3.1)	16% (F-Measure)
This study	**Security related and traditional software metrics from C/C++ projects**	**Modified GA**	**RF (cross-project)**	**88.8% 87.1% 83.3%**
Hin et al. [28]	Graph-based Features and Text mining features from C/C++ Systems	LineVD framework leveraging on control and data dependencies between statements	Graph Neural Networks (cross-project)	21% 50% 30%
Medeiros, Ivaki, Costa and Vieira [227]	Traditional software metric-based from C/C++ Systems	Dimensional reduction: Correlation and redundancy analysis. Application scenario	Random Forest, Extreme Boosting (cross-project)	(Recall) 90% (F-measure) 32 Precision will be low since f-measure is low

		criticality definition		
Le, Babar and Thai [30]	Source code of functions of Kotlin, Swift, Rust and C/C++ Systems	Self attention mechanism	Transformer model	47.9%, 69.2% 63.3%

4.5.2.1 Discussion

As can be seen in Table 6, the performance of software VPMs in this study were consistent and more than the performance in the stated recent works, in both within-project and cross-project predictions. Performance in the recent studies showed inconsistencies in precision and recall measures. This implies the software VPMs' results have the flaws of either high false positives (in case of high recall but low precision) or high false negatives (in case of high precision but low recall).

4.6 Threats to Validity

This study faces threats to construct validity, internal validity, and external validity. Construct validity is addressed by using multiple static code analyzers and labeling source files based on reported vulnerability cases. Internal validity is maintained through the causal relationship between the security-related metrics and vulnerabilities. However, external validity is limited as the models are specific to C/C++ projects and may not generalize to other languages. The study's scope excludes projects written in languages like Python, JavaScript, and C#.

5. Conclusion

This study developed realistic software security VPMs by leveraging relevant feature creation and optimization technique. Adopting these models for C/C++ projects can reduce data breaches and various security vulnerabilities, minimize the cost of fixing vulnerabilities, and prioritize limited resources for vulnerability inspection. The inclusion of security-related metrics in the training dataset enhances the predictive capacity of the models. Oversampling the imbalanced feature instances improves predictability. Optimizing features and mitigating local optima in the GA contribute to the models' performance in cross-project prediction.

Recommendations include using the developed models in conjunction with static analysis tools for comprehensive vulnerability detection and patching. The models complement static analysis tools by mitigating false positives and false negatives, and capturing of implicit vulnerabilities. Future work could involve extending the models to other programming languages like Python, JavaScript, and C#, improving cross-project performance, and exploring innovative feature selection techniques and classifiers for enhanced performance.

References

[1] Pereira, J. D. A., Ivaki, N., & Vieira, M. (2021). Characterizing buffer overflow vulnerabilities in large C/C++ projects. *IEEE Access, 9,* 142879–142892.

[2] Cao, S., Sun, X., Bo, L., Wu, R., Li, B., Wu, X., & Liu, W. (2023). Learning to detect memory-related vulnerabilities. *ACM Transactions on Software Engineering and Methodology*, *33*(2), 1–35.

[3] Siavvas, M., Gelenbe, E., Kehagias, D., & Tzovaras, D. (2018). Static analysis-based approaches for secure software development. In *Security in Computer and Information Sciences: First International ISCIS Security Workshop 2018, Euro-CYBERSEC 2018, London, UK, February 26-27, 2018, Revised Selected Papers* (pp. 142–157). Springer International Publishing.

[4] Khare, A., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., & Naik, M. (2023). Understanding the effectiveness of large language models in detecting security vulnerabilities. *arXiv preprint arXiv:2311.16169.*

[5] Marashdih, A. W., Zaaba, Z. F., Suwais, K., & Mohd, N. A. (2019). Web application security: An investigation on static analysis with other algorithms to detect cross site

scripting. *Procedia Computer Science, 161*, 1173–1181.

[6] Kaya, A., Keceli, A. S., Catal, C., & Tekinerdogan, B. (2019). The impact of feature types, classifiers, and data balancing techniques on software vulnerability prediction models, *Journal of Software: Evolution and Process, 31*(9), e2164.

[7] Li, Z., & Shao, Y. (2019). A survey of feature selection for vulnerability prediction using feature-based machine learning. In *Proceedings of the 2019 11th International Conference on Machine Learning and Computing* (pp. 36–42).

[8] Aruna, E. R., Mohan Reddy, A. R., & Sunitha, K. V. N. (2022). Secure SDLC using security patterns 2.0. In *IoT with Smart Systems: Proceedings of ICTIS 2021, Volume 2* (pp. 699– 708). Springer Singapore.

[9] Khan, S. A., Kumar, R., & Khan, R. A. (2023). *Software security: Concepts & practices*. Chapman and Hall/CRC.

[10] 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.

[11] Zimmermann, T., Nagappan, N., & Williams, L. (2010). Searching for a needle in a haystack: Predicting security vulnerabilities for Windows Vista. In *2010 Third International Conference on Software Testing, Verification and Validation* (pp. 421–428). IEEE.

[12] Morrison, P., Herzig, K., Murphy, B., & Williams, L. (2015). Challenges with applying vulnerability prediction models. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security* (pp. 1–9).

[13] Alves, H., Fonseca, B., & Antunes, N. (2016). Experimenting machine learning techniques to predict vulnerabilities. In *2016 Seventh Latin-American Symposium on Dependable Computing* (pp. 151–156). IEEE.

[14] Jimenez, M. (2018). *Evaluating vulnerability prediction models* (Doctoral dissertation). University of Luxembourg, Luxembourg.

[15] Shin, Y., Meneely, A., Williams, L., & Osborne, J. A. (2016). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering, 37*(6), 772–787.

[16] Filus, K., Boryszko, P., Domańska, J., Siavvas, M., & Gelenbe, E. (2021). Efficient feature selection for static analysis vulnerability prediction. *Sensors, 21*(4), 1133.

[17] Zhang, J., Wang, C., Li, A., Sun, W., Zhang, C., Ma, W., & Liu, Y. (2024). An empirical study of automated vulnerability localization with large language models. *arXiv preprint arXiv:2404.00287*.

[18] Kile, A., Garba, M., & Bassi, A. (2025). Identi-fix: An SVM-based automated software vulnerability scanner for C/C++ projects. *International Journal of Cybersecurity and Software Engineering, 8*(2), 112–128.

[19] Ahmed, H., Harzevili, A., Shin, J., Pham, T., & Wang, Y. (2025). SECVULEVAL: A benchmark for evaluating large language models in fine-grained C/C++ detection. *Journal of Software Security Research, 12*(3), 45–67.

[20] Huynh, L., Zhang, Y., Jayasundera, D., Jeon, W., Kim, H., Bi, T., & Hong, J. B. (2025). Detecting code vulnerabilities using large language models: Evaluation and correction strategies. The University of Western Australia/Sungkyunkwan University Research Paper.

[21] Grahn, D., & Zhang, J. (2021). An analysis of C/C++ datasets for machine learning-assisted software vulnerability detection. In *Proceedings of the Conference on Applied Machine Learning for Information Security*.

[22] Joseph, V. R. (2022). Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal, 15*, 531–538.

[23] Malhotra, R., Khanna, M., & Raje, R. R (2016). On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions. *Swarm and Evolutionary Computation, 32*, 85–109.

[24] Park, J., Shin, J., & Choi, B. (2023). Reduction of false positives for runtime errors in C/C++ software: A comparative study. *Electronics, 12*(16), 3518.

[25] Tanwar, A., Sundaresan, K., Ashwath, P., Ganesan, P., Chandrasekaran, S. K., & Ravi, S. (2020). Predicting vulnerability in large

codebases with deep code representation. *arXiv preprint arXiv:2004.12783*.

[26] Harman, M., Jia, Y., & Zhang, Y. (2015). Achievements, open problems and challenges for search-based software testing. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation* (pp. 1–12). IEEE.

[27] Medeiros, N., Ivaki, N., Costa, P., & Vieira, M. (2020). Vulnerable code detection using software metrics and machine learning. *IEEE Access, 8*, 219174–219198.

[28] Hin, D., Kan, A., Chen, H., & Babar, M. A. (2022). Line VD: Statement-level vulnerability detection using graph neural networks. In *Proceedings of the 19th International Conference on Mining Software Repositories* (pp. 596–607).

[29] Kalouptsoglou, I., Siavvas, M., Kehagias, D., Chatzigeorgiou, A., & Ampatzoglou, A. (2022). Examining the capacity of text mining and software metrics in vulnerability prediction. *Entropy, 24*(5), 651.

[30] Le, T. H., Babar, M. A., & Thai, T. H. (2024). Software vulnerability prediction in low-resource languages: An empirical study of CodeBERT and ChatGPT. *arXiv preprint arXiv:2404.17110*.