Comparing the Capabilities of Ensemble Learning Algorithms and SAST Tools for Effective Code Based Vulnerability Detection

MSc Research Project
MSc in Cybersecurity

Ashwathy Ajaykumar Marath
Student ID: x23166371

School of Computing
National College of Ireland

Supervisor: Prof. Arghir Nicolae Moldovan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ashwathy Ajaykumar Marath		
Student ID:	X23166371		
Programme:	MSc in Cybersecurity	**Year:**	2024-2025
Module:	MSc Research Project		
Supervisor:	Prof. Arghir Nicolae Moldovan		
Submission Due Date:	12-12-2024		
Project Title:	Comparing the Capabilities of Ensemble Learning Algorithms and SAST Tools for Effective Code Based Vulnerability Detection		
Word Count:	7146	**Page Count:**	20

Signature: Ashwathy Ajaykumar Marath

Date: 12-12-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Comparing the Capabilities of Ensemble Learning Algorithms and SAST Tools for Effective Code Based Vulnerability Detection

Ashwathy Ajaykumar Marath
X23166371

Abstract

Based on the VUDENC and DiverseVul benchmarks, this work evaluates ensemble learning algorithms and SAST tools for software vulnerability detection. Secondary qualitative research data was collected between 2016 and 2024, and quantitative experiments were employed. For handling class imbalance, both Random Forest, XGBoost, LightGBM, and CatBoost ensemble models were experimented on with and without SMOTE. Ensemble models perform better than SAST techniques with XGBoost having the highest ROC-AUC score of 0.76 and Random Forest having stable majority class accuracy. SAST tools were okay for level L concerns but had higher levels of false positives and lower precision. Hybrid techniques can be used in the future to minimize false alarms and enhance immunity to software attacks in ensemble models.

1 Introduction

As a result of the increase in cyber risks and other related threats by code vulnerabilities, software security has become more important. It is essential to identify these weaknesses during execution of the software development life cycle. Although static technologies like SAST are common, they have pre-identified patterns whereby challenges like high false positive and false negative rates are common. Therefore, only advanced methods should enhance and complement their performance (Baptista et al., 2021). Appealing possibilities are presented by the potential of machine learning and more specifically by improvements in ensemble learning algorithms concerning complicated data and forecast accuracy. A lot of industries have endorsed techniques such as XGBoost, Decision Trees, and Random Forests (Bhardwaj, 2022). These algorithms need to be advanced more now and to do so, methodologies such as (SMOTE, for instance) must be used when dealing with unbalanced datasets.

1.1 Background

Software security and reliability are all about identifying the vulnerable areas in the code but the efficacy of traditional Static Analysis Application Security Testing approaches, which are still popular, is relatively low because of problems like high false positive and false negative ratios (Galinde, 2023). The performance of SAST tools on the VUDENC dataset, as well as ensemble learning approaches on the DiverseVul dataset, are compared in this study. To measure the degree to which the models intended to identify vulnerabilities, ensemble methods like Decision Trees, Random Forest, XGBoost, LightGBM, and CatBoost were applied. To compare, machine learning models were trained to make more accurate predictions about the existence of novel susceptibilities by employing the Synthetic Minority Oversampling Technique (SMOTE) to balance the class distribution of the DiverseVul dataset. Ensemble learning may aid code-based vulnerability detection systems by augmenting other SAST approaches. This research has sought to provide useful information about this topic by comparing the various methods in question with the help of media such as recall, accuracy, precision, and ROC-AUC.

1.2 Motivation

This research is being undertaken because there is a critical requirement to improve software vulnerability detection systems due to increased cyber-attacks. Even though SAST technologies had mass usage some time back they were largely ineffective due to high false positive and negative rates. However, new ways of safer and more reliable detection have been made possible due to the advancement of artificial intelligence technologies, especially ensemble learning methods. The project's objective is to integrate these methodologies as well as novel approaches such as SMOTE to strengthen vulnerability detection systems. The end goal is to contribute to reducing the risks involved in software development processes and improve the state of the art of cybersecurity frameworks by managing imbalances in the data.

1.3 Research Aim and Objectives

This study aims to compare ensemble learning algorithms and SAST tools for code-based vulnerability detection, focusing on improving accuracy, reducing false positives and negatives, and providing insights into optimizing detection systems through a comprehensive evaluation of their capabilities

- To evaluate the performance of ensemble learning algorithms, including Decision Trees, Random Forest, XGBoost, LightGBM, and CatBoost, in detecting software vulnerabilities using the DiverseVul dataset.
- To analyze the effectiveness of SAST tools in identifying code vulnerabilities using the VUDENC dataset, focusing on accuracy and reliability.
- To compare the strengths and limitations of ensemble learning techniques and SAST tools, providing insights for optimizing vulnerability detection systems.

1.4 Research Question

How do ensemble learning algorithms compare to Static Application Security Testing (SAST) tools in terms of accuracy, reliability, and reducing false positives and false negatives for code-based vulnerability detection?

1.5 Contribution

- The main contribution of this study is to generate an in-depth knowledge of software security, weaknesses in the code are also sought by employing ensemble learning techniques in contrast with SAST.

- This emphasizes ensemble techniques such as Random Forest, XGBoost, Decision Tree, LightGBM, and CatBoost can complement traditional SAST tools to enhance identification rates and address underrepresented classes through SMOTE.
- The study tests the techniques on several data sets and indicates the benefits and shortcomings of the different approaches applied to antibiotics such as DiverseVul and VUDENC, thus aiding the enhancement of vulnerability detection systems.
- This will make it possible to develop more effective strategies for improving the security of software systems.

2 Related Work

2.1 Overview

Tracking vulnerabilities that have been flagged by security scanning software takes a lot of time, more so in large, advanced communication networks. In that sense, the impact that different software vulnerabilities can have on a given IT system can vary depending on the environment (Siewruk, G. and Mazurczyk, 2021). The number of faults reported by scanners can run into thousands making research and setting levels of priority rather expensive for organizations. The context-aware software vulnerability categorization method, referred to as Mixeway, seeks to streamline this process with the use of machine learning. For example, we show that once the selected parts of a deep learning network are frozen, a training procedure performed with other known and evaluated vulnerabilities enables predicting the severity class of a newly discovered vulnerability if the description is adequately processed using the Natural Language Processing methods (Siewruk, G. and Mazurczyk, 2021). The results of experiments on the 12-month collected dataset of one of the largest Polish mobile network operators do suggest that the Mixeway approach is effective and beneficial.

The software business and cybersecurity community have observed that the growth of disclosed security defects indicates that vulnerability discovery techniques require improvement. The open-source community has emerged in the software domain, there exist voluminous software codes for learning by machines and data mining (Lin *et al.*, 2020). Deep learning has recently been implemented in speech recognition and machine translation applications which implies the neural models have a natural language comprehension ability. This has led to the use of deep learning by software engineers alongside cybersecurity engineers to learn some of the patterns and semantics of vulnerable code (Lin *et al.*, 2020). The study reviews literature on software vulnerability detection using deep learning or neural networks to understand how the current research applies neural approaches to code semantics for vulnerability identification. There are the challenges of this new field and potential search strategies identified here.

2.2 Static Application Security Testing (SAST) Tools: Strengths, Limitations, and Use Cases

There has been a rise recently in smart contract attacks, and therefore security has become vital. To solve this, several approaches in SAST methods have been suggested for improving smart contract vulnerability detection. It is not easy to compare these instruments logically to consider their effectiveness (Li *et al.*, 2024). To address this gap, the authors advance an improved, high-level classification of 45 smart contract vulnerability types in this study. The authors give an overview that contains 40 types of code and various characteristics and patterns of DVCs and applications using this as a reference. In evaluating these 8 SAST tools in line with this benchmark of 788 smart contract files, and 10,394 vulnerabilities, the authors came up with the following. The findings provide evidence that conventional SAST instruments fail to detect as many as 50% of the benchmark flaws, combined with high levels of imprecision and false positives that equal 10% (Li *et al.*, 2024). This study also establishes the use of various techniques as decreasing the false negative rate but identifies 36.77 percentage points more functions as questionable.

Static code analysis or source code analysis is important to software development and one of the most important dimensions of an application is static application security testing. Nevertheless, a comparison of SAST tools to select the appropriate tools for identifying vulnerabilities is challenging (Li *et al.*, 2023). From 161 available SAST tools, following criteria we selected exemplar 7 free or open-source tools considered in this work. We compared these SAST tools based on their effectiveness, reliability, and efficiency with synthetic and recently invented realistic benchmarks. Tools for SAST achieve good performances on synthetic datasets, but only 12.7% of realistic weaknesses (Li *et al.*, 2023). The detection capabilities of the tools rise to 70.9% of all vulnerability's unseen, and most importantly different from resource management vulnerabilities and somewhat poorly handled input/output vulnerabilities. They built the detecting criteria and incorporated them into abilities; however, the detection result was not satisfactory.

2.3 Ensemble Learning Algorithms for Software Vulnerability Detection

Preventative strategies and techniques utilized during software development help identify many vulnerable software parts at an early stage to reduce testing costs and produce a dependable and robust software system. Past studies have indicated that intelligent prediction methods might reveal weaknesses of the system, but the lack of enough training datasets limits their application (Pang *et al.*, 2016). This research provides a method for early prediction of software component vulnerability. The proposed method refers to potentially susceptible components as mislabelled data that may contain actual, though disguised, vulnerabilities. The ensemble learning and support vector machine algorithm is incorporated into the hybrid method to identify the susceptible components. The proposed vulnerability detection system is evaluated and considered against the background of Java Android apps (Pang *et al.*, 2016). From the empirical evaluation, it was found that the proposed hybrid method was effective in identifying susceptible classes with high precision and acceptable accuracy as well as recall.

One of the greatest emerging software security issues is the use of automated tools for vulnerability identification. It is possible to automate a process of extracting vulnerabilities with the help of deep learning to a certain extent (Wang *et al.*, 2020). This work employs the deep representation learning method and heterogeneous ensemble learning to identify these vulnerabilities smartly and independently. Only vulnerability portions of the source code are used in the trials first, original code files are pre-processed to reduce code analysis and increase detectability. Second, the authors focus on the corpus with pre-training and retain semantic features to express data samples as vectors. Third, the vectors are processed via a deep-learning model to identify if a device is vulnerable (Wang *et al.*, 2020). Lastly, the authors obtain multiple classifiers for each case and train homogenous and heterogenous ensemble classifiers. In evaluating the detection approach, we therefore compare the efficiency and resource utilization of both the network models, the pre-training techniques, classifiers, and vulnerabilities. The experimental observation indicates our technique enhanced false positive, false negative, accuracy, recall, and F1 Features.

2.4 Comparative Studies on Machine Learning and Traditional Tools for Code Security

Source code static analysis is used by software developers to identify weaknesses. Human-expert vulnerability patterns are difficult and time-consuming, and therefore, they must rely on machine learning to identify vulnerability (Li *et al.*, 2019). Current research proposes employing deep learning to detect vulnerabilities with little to no need for professionals to define rules or traits. As with vulnerability detection, there is no clear understanding of how the various factors influence it. This research employs two datasets of two programs with 126 types of vulnerability for the first comparison analysis to determine the impact of various parameters on vulnerability detection (Li *et al.*, 2019). The results also show that the accommodation of control dependence helps to enhance vulnerability detection F1-measure by 20.3%, and there is no significant difference between the unbalanced data processing approaches for the datasets. Using the last output corresponding to the time step for the bidirectional long short-term memory (BLSTM) can decrease the false negative rate by 2.0%, at the same time it increases false positives by 0.5%.

Due to these techniques' development in the broader field of machine learning, academics have attempted to use the same in many software engineering activities that involve source code analysis including testing and identification of vulnerabilities (Sharma *et al.*, 2021). Due to many publications, it can be challenging for people to understand the scientific environment. It provides a state-of-the-art understanding of applied machine learning for source code analysis. A discussion of the machine learning techniques, resources, and applied methodologies for addressing twelve problems in software engineering is presented. Based on the database literature search the authors get 479 main study collecting data from the period of 2011 to 2021 (Sharma *et al.*, 2021). The listed studies enable us to draw conclusions and summarise our observations and findings. The author's investigation showed that machine learning is being employed for source code analyses.

2.5 Addressing Class Imbalance in Machine Learning for Vulnerability Detection

There is minimum empirical literature on SDD which is a binary classifier problem having skewed data distribution and tilted learning in favor of one class (Kim and Chung, 2024). The richness of information is maintained by deep learning and machine learning to employ four class balancing methods including SMOTE, ADASYN, SMOTE-Tomek, and SMOTE-ENN for the SDD problem. Deep learning employs MLP, CNN, and LSTM and machine learning employs decision trees, random forest, logistic regression, and XGB (Kim and Chung, 2024). Therefore, we evaluate and discuss class balancing techniques on those models. Our investigation revealed that the applications of class balancing strategies were positive on MLP, negative on CNN and LSTM, and positive on all machine learning techniques.

2.6 Summary

Study (Author, Year)	Purpose	Key Features	Strengths	Weaknesses	Proposed Algorithm/Approach
Siewruk and Mazurczyk (2021)	To streamline vulnerability tracking using machine learning in large communication networks.	Introduced Mixeway, a context-aware software categorization method using NLP and deep learning.	Demonstrated effectiveness in predicting severity class with a 12-month dataset from a Polish mobile operator.	Requires adequately processed vulnerability descriptions and is environment dependent.	Mixeway method with selected frozen layers and NLP for severity prediction.
Lin et al. (2020)	To review software vulnerability detection using deep learning and neural networks.	Neural models comprehend code semantics, leveraging deep learning for vulnerability identification.	Highlights the potential of deep learning for pattern recognition and vulnerability detection in software code.	Identifies challenges and gaps in applying neural models to code semantics.	Use of deep learning to capture patterns and semantics of vulnerable code.

Li et al. (2024)	To classify and evaluate SAST tools for detecting smart contract vulnerabilities.	Evaluated 8 SAST tools on 788 smart contract files and 10,394 vulnerabilities	Identified high false positives (10%) and evidence that SAST tools miss 50% of benchmark flaws.	SAST tools performed inconsistently and failed to detect a significant portion of vulnerabilities	High-level classification of 45 smart contract vulnerabilities to refine detection benchmarks.
Li et al. (2023)	To assess the effectiveness of open source SAST tools in detecting realistic vulnerabilities	Compared to 7 free/open-source tools using synthetic and realistic benchmarks	Improved detection capabilities on unseen vulnerabilities (70.9%) with 12.7% success on realistic weaknesses.	Poor performance in handling input/output vulnerabilities and resource management flaws.	Developed detection criteria and incorporated them into SAST tools
Pang et al. (2016)	To propose a hybrid method for early prediction of software component vulnerabilities.	Combined ensemble learning and SVM to identify susceptible components.	Achieved high precision, accuracy, and recall in identifying vulnerable classes in Java Android apps.	The lack of sufficient training datasets limits broader applications.	Hybrid approach combining ensemble learning and SVM for early vulnerability detection.
Wang et al. (2020)	To automate vulnerability detection using deep representation and ensemble learning methods.	Pre-processed original code, retained semantic features, and used ensemble classifiers.	Enhanced false positive, and false negative rates, and improved accuracy, recall, and F1 score in experimental observations.	Resource-intensive due to pre-training techniques and large-scale trials.	Heterogeneous and homogeneous ensemble classifiers for automated vulnerability detection.
Li et al. (2019)	To employ deep learning for vulnerability detection with minimal human intervention.	BLSTM model with control dependence for analyzing code and datasets.	Improved the F1 measure by 20.3% and decreased false negatives by 2.0%.	Slight increase in false positives (0.5%) and limited dataset scope for evaluation.	Bidirectional long short-term memory (BLSTM) with control dependence for enhanced detection.
Sharma et al. (2021)	To provide an overview of machine learning techniques for source code analysis and	Reviewed 479 studies on ML techniques applied to software engineering	Comprehensive overview of applied machine learning in source code analysis,	Generalized findings may not apply directly to specific use cases or tools.	Summary of state-of-the-art machine learning techniques for source code analysis.

	vulnerability detection.	problems.	summarizing trends and methodologies.		
Kim and Chung (2024)	To address the class imbalance in software defect detection (SDD) problems using class balancing	Evaluated SMOTE, ADASYN, SMOTE-Tomek, and SMOTE-ENN on various deep learning & ML models	Positive impact on MLP, decision tree, random forest, and XGB models, with improved richness of info for imbalanced datasets	Negative impact on CNN and LSTM models, highlighting inconsistencies across different techniques	Class balancing techniques applied to MLP, CNN, LSTM, and various ML models for SDD.

3 Research Methodology

Using both qualitative and quantitative methodologies this paper uses an assessment of software vulnerability identification employing SAST tools and ensemble learning algorithms. The application includes data exploration, analysis, and interpretation by following the Knowledge Discovery in Databases (KDD) process. The papers selected are from 2016–2024 literature review forms the basis of the qualitative component of the study. This study describes the SAST tool, ensemble learning, and software vulnerability detection development. The summary table in the literature review provided an aggregate perspective of their strengths, weaknesses, and trends based on the information in scholarly publications, technical reports, and case studies. The qualitative study gives the theoretical background for the comparison of the classical SAST methods and Ensemble Learning. Specifically, for the quantitative evaluation of SAST tools and ensemble learning approaches, the VUDENC and DiverseVul datasets are utilized. The dataset DiverseVul was analyzed with Decision Trees, Random Forests, XGBoost, LightGBM, and Cat Boost ensemble learning models. DiverseVul employed SMOTE to enhance model detection because of the issue of class imbalance. It is analyzed for accuracy, precision, recall, F1 score, and ROC AUC and both methods were compared. The paper comparing SAST tools and ensemble learning algorithms is based on a qualitative review of the related literature as well as a quantitative analysis of the obtained datasets. The integration of qualitative and quantitative approaches provides a multidimensional investigation making the vulnerability detection systems more credible.

3.1 KDD Framework

Knowledge Discovery in Databases, or KDD, is a systematic technique for deriving useful lessons and patterns from large sets of data. Data selection, pre-processing, transformation, data mining, interpretation, or evaluation are processes involved in this procedure. The KDD framework was chosen because it offers a structured approach to handling complex datasets, thus ensuring robust pre-processing and evaluation workflows. Other frameworks, such as CRISP-DM or SEMMA, were considered, but the iterative process of KDD fit better with the dynamic nature of the tasks related to vulnerability detection.

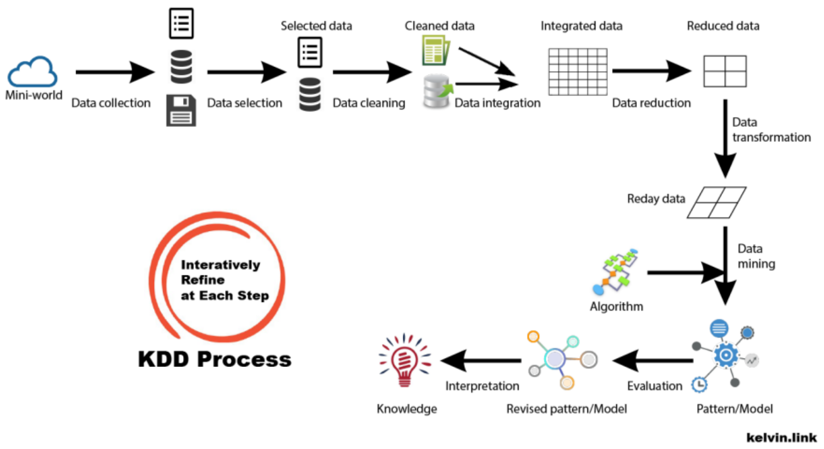


Figure 1: KDD Framework. Source: (Chehab, 2020)

Data Selection: This study used the VUDENC dataset because of its comprehensive labeling of vulnerabilities and good alignment with SAST tools. DiverseVul is selected because of the diversity of its vulnerabilities, which provides good training of machine learning models. As compared to the Juliet Test Suite or CodeXGLUE, these datasets gave richer and more applicable benchmarks to this study.

Data Pre-processing: Concerning the previous case, additional data were added to fill the empty spaces aiming to increase the effectiveness of the model. A standard preprocessing method was used to enable relevant conclusions to be drawn from the output of SAST tools before feeding them to VUDENC.

Data Transformation: To convert the textual information into numerical information which can be utilized by ensemble methods, feature extraction of the DiverseVul dataset was performed by using TF-IDF (term frequency-inverse document frequency). Scale features were split into a training set, which is used to fit the data, and a test set which is used to assess performance for statistical analysis.

Data Mining: DiverseVul dataset was used to train and test Decision Trees, Random Forest, XGBoost, LightGBM, and CatBoost ensemble learning algorithms. Recall that VUDENC was used to test SAST vulnerability detection tools. Accuracy, precision, recall, and ROC-AUC statistics were computed for performance evaluation.

Interpretation and Evaluation: Utilization of an ensemble learning algorithm in conjunction with the SAST tool results dynamics enabled insight into the comparative advantages and weaknesses of the two variables. This research demonstrates the possibility of enhancing vulnerability detection systems based on SAST techniques by using ensemble methods.

3.2 Libraries Imported

The project required data processing, machine learning, and assessment of Python packages. Pandas allows structure data manipulation and JSON-loaded datasets. Visualization tasks drew graphs and distributions with the use of Matplotlib. The SMOTE was chosen instead of other techniques like ADASYN since this method generates synthetic samples without losing the general distribution of the data. Thus, it can be very effective for handling the class imbalance found in DiverseVul. Other techniques like SMOTE-ENN were not used here since these require more computational overhead with a greater risk of overfitting. The function of TfidfVectorizer from sklearn.feature_extraction.text is to summarize text for the feature. Ensemble techniques were used using machine learning models such as DecisionTreeClassifier, RandomForestClassifier, XGBClassifier, LGBMClassifier, and CatBoostClassifier from sklearn, xgboost, lightgbm, and catboost. When the machine learning model's performance was evaluated using metrics obtained from scikit-learn's metrics module, the obtained accuracy, ROC-AUC score, as well as a classification report. The train_test_split of the sklearn.model_selection package separated the data into a training and a testing set. During dataset preparation, attempts at modeling; and analysis, these packages provided a good platform.

3.3 Feature Extraction

This research converted unstructured text into numerical features suitable for use in machine learning models. Using the TfidfVectorizer tool, the func column of the DiverseVul dataset was transformed into a TF-IDF matrix. TF-IDF was used because it is simple and efficient for text-based datasets. Methods like Word2Vec or BERT capture contextual information of the words; however, TF-IDF was more computationally efficient for the goals in this study. This approach enhanced the performance of vulnerability detection by training systems based on ensembling learning using textual information.

3.4 Data Split

In this research, the dataset was separated for training and testing machine learning models. The dataset was partitioned 80:20 to form the training and the testing sets using Train_test_split from sklearn.model_selection. This approach enabled models to learn from 80 percent of the data and use the remaining 20 percent to test their performance. The use of this splitting method maintained the integrity of the class distributions so that the training and testing datasets were representative of the primary dataset. In this respect, these methods evaluated models and thus provided a means for more detailed comparisons of the 12-vulnerability detection and overfitting/bias reduction capabilities of ensemble learning algorithms.

3.5 Dataset Description

This study worked on VUDENC and DiverseVul, two datasets that are devoted to individual analysis (LauraWartschinski, 2019; surrealyz, 2024). The VUDENC dataset used in the experiments is comprised of labeled vulnerabilities of different software systems which were used to evaluate the effectiveness of the SAST tools in terms of accuracy, precision, and reliability. The DiverseVul project has sufficient detailed information on the functional behavior of the source code and vulnerabilities which is applicable for machine learning. To deal with the class imbalance problem in this dataset, SMOTE feature extraction and balancing were applied. These datasets permitted extensive research of the traditional SAST approaches as well as the relatively recent combined learning techniques for the search of vulnerabilities.

3.6 Justification of SAST and Ensemble Learning Algorithms

This research used SAST tools and a collection of learning methods due to their combined strengths in the identification of vulnerabilities. SAST tools are mostly employed in static code analysis revealing vulnerabilities as the software is being developed. However, their highly high false positive and negative rates necessitate new solutions. Ensemble learning methods such as Random Forest, Decision Tree, XGBoost, LGBM, and CatBoost improve accuracy and work well with complex datasets. This work integrates the rule-based detection capabilities of SAST tools with the data-driven insights of ensemble models to increase the accuracy, reliability, and resiliency performance of code-based vulnerability detection systems.

4 Design Specification

This study systematically assesses the SAST tools and Ensemble Learning approaches for the identification of vulnerabilities. The data preparation includes loading VUDENC and DiverseVul followed by SMOTE imbalance correction and TFIDF feature extraction. The data is split into training and testing sets for model training. The study employed Random Forest, XGBoost, LightGBM, and CatBoost ensemble learning methods. Evaluation measures include accuracy, precision, recall, F1-score, and ROC-AUC.

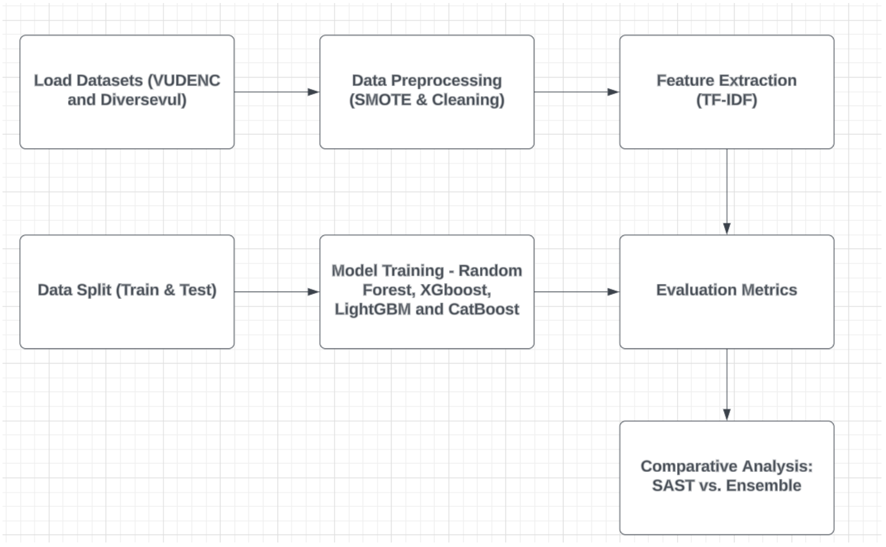

Load Datasets (VUDENC and Diversevul)
Data Preprocessing (SMOTE & Cleaning)
Feature Extraction (TF-IDF)
Data Split (Train & Test)
Model Training - Random Forest, XGboost, LightGBM and CatBoost
Evaluation Metrics
Comparative Analysis: SAST vs. Ensemble

Figure 2: Workflow Diagram

5 Implementation

- Python and a host of libraries were employed. Data processing was done with the aid of Pandas which ensured that the initial data received was structured in a consistent manner for analysis. Initially, VUDENC and DiverseVul were imported separately. They were then cleaned and merged into a single DataFrame with a modeling traintest proportion of 80:20.

- SAST and machine learning were integrated to detect and capture vulnerabilities. Bandit and SonarQube SASTs were used to assess the presence of codes that were a cause of concern. Among the challenges addressed were severity and confidence issues within the source code.
- Some of the machine learning models applied included Decision Trees, Random Forest, XGBoost, and CatBoost as well as LightGBM. In the case of class imbalance, the SMOTE technique was utilized on the preprocessed data. Numbers for feature extractions were generated from text through the TF-IDF vectorizer.
- Grid and random search were the approaches used in fine tuning the models to improve accuracy. Evaluation metrics that were deployed include measures such as ROC-AUC values, precision, recall and F1-score.
- The comparison sought to demonstrate how SAST tools stack up against ensemble learning techniques in locating code vulnerabilities while minimizing false positive and negatives.
- Hardware: RAM – 8GB, Operating System – Windows 11, Processor – i5

6 Evaluation

6.1 EDA – Exploratory Data Analysis

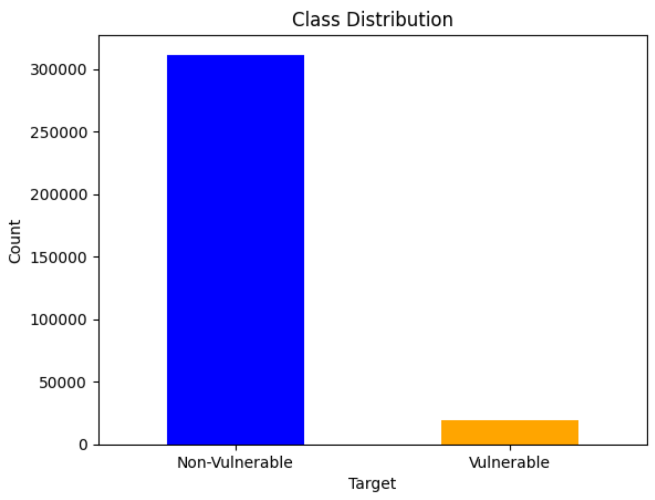


Figure 3: Class Distribution

As the class distribution graph indicates, the dataset has more instances of non-vulnerable instances than vulnerable ones. This disparity means that the model training must incorporate oversampling, under sampling, or appropriate techniques so that bias does not occur, and the right predictions are given.

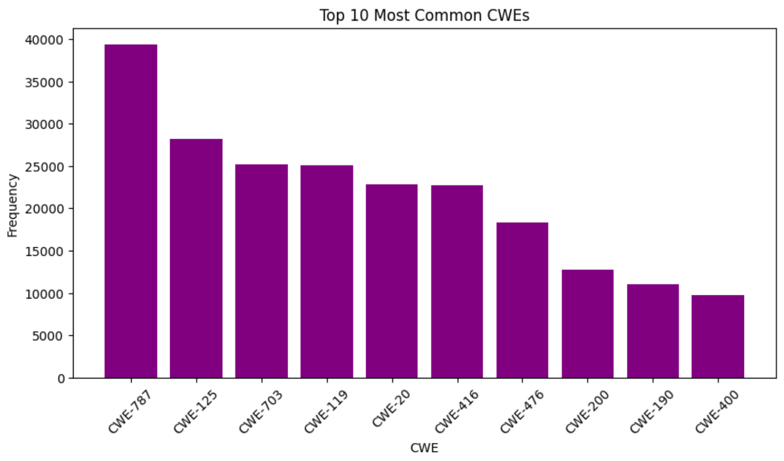


Figure 4: Top 10 Most Common CWEs

CWE-787 (Out-of-Bounds Write) is the most common in the chart above. This highlights the important issues which must be tackled by developers as well as security teams. The frequency distribution supports the aim of focusing on certain classes of CWE for vulnerability and software security enhancement.

6.2 Decision Tree Classifier

Table 1: Result of Decision Tree Classifier

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.94	0.71	0.94	0.83	0.93	0.68
Recall	1	0.02		0.51	0.94	
F1-Score	0.97	0.03		0.5	0.92	
Support (Samples)	62,310	3,789	66099	66099	66099	

In Table 1, the modified Decision Tree yields a high accuracy of 94% and precision for non-vulnerable samples (class 0) but performs very poorly on vulnerable samples, class 1, with low recall of 0.02 and F1 score of 0.03. The macro F1 score is 0.50, showing uneven performance, while the weighted average F1 score is 0.92, reflecting class imbalance. The ROC-AUC is 0.68, which suggests that there is room for improvement by balancing classes or features.

6.3 Random Forest

Table 2: Result of Random Forest Classifier

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.95	0.2	0.92	0.57	0.9	0.73
Recall	0.97	0.12		0.54	0.92	
F1-Score	0.96	0.15		0.55	0.91	
Support (Samples)	62,310	3,789	66099	66099	66099	

In Table 2, the Random Forest model yielded 92% accuracy, with 0.95 precision and 0.97 recall for the non-vulnerable samples of Class 0, showing very low recall for the vulnerable samples of Class 1 at 0.12 and an F1 score of 0.15. The weighted F1 score is 0.91, reflecting the dominance of Class 0. A ROC-AUC of 0.73 reflects fair but improvable discriminative ability.

6.4 XGB Classifier

Table 3: Result of XGB Classifier

Metric	**Class 0 (non-vulnerable)**	**Class 1 (vulnerable)**	**Accuracy**	**Macro Avg**	**Weighted Avg**	**ROC-AUC Score**
Precision	0.94	0.47	0.94	0.71	0.92	0.78
Recall	1	0.04		0.52	0.94	
F1-Score	0.97	0.08		0.53	0.92	
Support (Samples)	62,310	3,789	66099	66099	66099	

In Table 3, it gives an accuracy of 94% in the XGBoost model, where recall for Class 0 (not vulnerable) was 1.00, and the F1 score was at 0.97, while for Class 1, representing vulnerable ones, it was extremely low, 0.04 and 0.08, respectively. The macro metrics clearly indicate poor handling of the minority class, though at a ROC-AUC of 0.78, which was better compared to most models.

6.5 SOTA Implementation

6.5.1 LGBM Classifier

Table 4: Result of LGBM Classifier

Metric	**Class 0 (non-vulnerable)**	**Class 1 (vulnerable)**	**Accuracy**	**Macro Avg**	**Weighted Avg**	**ROC-AUC Score**
Precision	0.94	0.62	0.94	0.78	0.93	0.79
Recall	1	0.02		0.51	0.94	
F1-Score	0.97	0.05		0.51	0.92	
Support (Samples)	62,310	3,789	66099	66099	66099	

In Table 4, the LightGBM model yielded an accuracy of 94%, where Class 0 had perfect recall of 1.00 and an F1 score of 0.97, while Class 1 had very poor recall of 0.02 with an F1 score of 0.05, which shows class imbalance. However, the ROC-AUC score is 0.79, which was higher than earlier models.

6.5.2 CatBoost Classifier

Table 5: Result of CatBoost Classifier

Metric	**Class 0 (non-vulnerable)**	**Class 1 (vulnerable)**	**Accuracy**	**Macro Avg**	**Weighted Avg**	**ROC-AUC Score**
Precision	0.94	0.49	0.94	0.72	0.92	0.79
Recall	1	0.04		0.52	0.94	
F1-Score	0.97	0.08		0.53	0.92	
Support (Samples)	62,310	3,789	66099	66099	66099	

In Table 5, CatBoost demonstrates its full capability in detecting non-vulnerable samples (class 0) with a perfect recall of 1.00 and a good F1 score of 0.97. Overall accuracy performance is 94%. However, it has a hard time detecting the susceptible samples (Class 1) having a recall of 0.04 and a very low F1 score of 0.08, which points to poor performance in terms of handling the minority class.

6.6 ROC Curve

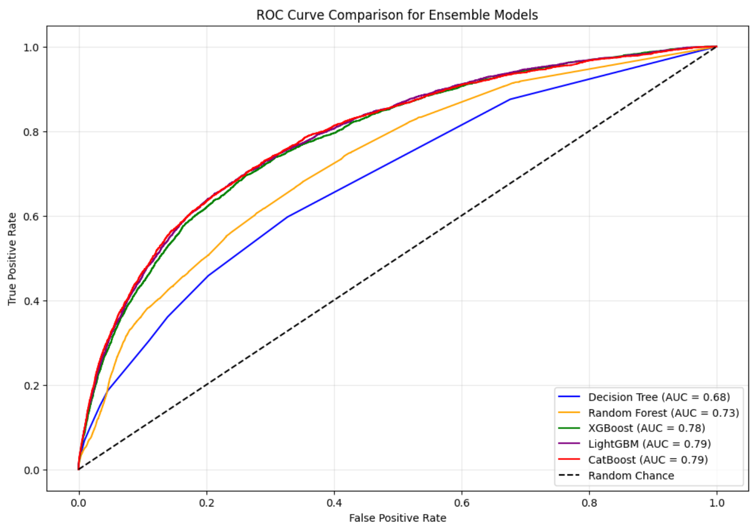


Figure 5: ROC Curve Comparison

The ROC curve comparison of ensemble models depicts their ability to classify susceptible and non-susceptible samples. The highest AUC values belonged to CatBoost and LightGBM at (0.79), indicating better discrimination. The AUC for Random Forest is good at 0.73 while XGBoost comes next with 0.78. Decision Tree had low AUC of 0.68 suggesting poor prediction. These curves show the degree to which the models can achieve high true positive rates without increasing false positives, where CatBoost and LightGBM were the best models. The results confirm the effectiveness of the new methods of ensemble-based vulnerability detection techniques.

6.7 Comparative Analysis of Ensemble Learning Algorithm without SMOTE

Table 6: Comparative Analysis of Ensemble Learning Algorithm without SMOTE

Metric	Decision Tree	Random Forest	XGBoost	LightGBM	CatBoost
Precision (Class 0)	0.94	0.95	0.94	0.94	0.94
Precision (Class 1)	0.71	0.2	0.47	0.62	0.49
Recall (Class 0)	1	0.97	1	1	1
Recall (Class 1)	0.02	0.12	0.04	0.02	0.04
F1-Score (Class 0)	0.97	0.96	0.97	0.97	0.97
F1-Score (Class 1)	0.03	0.15	0.08	0.05	0.08
Accuracy	0.94	0.92	0.94	0.94	0.94
Macro Avg Precision	0.83	0.57	0.71	0.78	0.72
Macro Avg Recall	0.51	0.54	0.52	0.51	0.52
Macro Avg F1-Score	0.5	0.55	0.53	0.51	0.53
Weighted Avg F1-Score	0.92	0.91	0.92	0.92	0.92
ROC-AUC Score	0.68	0.73	0.78	0.79	0.79

The ROC-AUC values of LightGBM and CatBoost were the highest (0.79), which means better performance. The precision and recall for Class 0 are good, with Class 1 having a more erratic performance, however. While Random Forest and XGBoost achieved similar accuracy, the recall for Class 1 was lower relative to the results for LightGBM and CatBoost. The Decision Tree has a low AUC value but is correct. Therefore, LightGBM and CatBoost seem to be performing well in terms of predictive power and discrimination.

6.8 Comparative Analysis of Ensemble Learning Algorithm with SMOTE

Table 7: Random Forest with SMOTE Results

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.95	0.17	0.9	0.56	0.91	0.72
Recall	0.94	0.19		0.57	0.9	
F1-Score	0.95	0.18		0.56	0.9	
Support	62,310	3,789	66,099	66,099	66,099	

In Table 7, Random Forest model using SMOTE performs with an accuracy of 90% with a ROC-AUC score of 0.72. Class 0 (non-vulnerable) successfully predicted non-vulnerable samples with a good level of accuracy (0.95), recall (0.94), and an F1 score of (0.95). SMOTE balancing helps but Class 1 (Vulnerable) still has a very low accuracy of 0.17 and recall of 0.19 which signifies challenges in detecting susceptible samples. To improve vulnerability detection these results, indicate that optimization is needed.

Table 8: Decision Tree with SMOTE Results

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.96	0.12	0.79	0.54	0.91	0.69
Recall	0.81	0.43		0.62	0.79	
F1-Score	0.88	0.19		0.53	0.84	
Support	62,310	3,789	66,099	66,099	66,099	

In Table 8, Decision Tree model, enhanced by SMOTE, reaches an accuracy of 79% with the ROC-AUC indicator at 0.69. However, Class 0 (Non-Vulnerable) managed to maintain relatively good accuracy 0.96 with an F1 score grading of 0.88 for identification of non-vulnerable samples. Class 1 (Vulnerable) had a fair F1 score of 0.19, a relatively lower accuracy of 0.12 with recall at 0.43. In practice, this means that despite SMOTE improving the recall over the susceptible sample, the overall performance indicates the challenge of balancing for class detection. More efforts must be applied in detecting Class 1 vulnerabilities.

Table 9: XGBoost with SMOTE Results

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.96	0.18	0.86	0.57	0.92	0.76
Recall	0.88	0.43		0.66	0.86	
F1-Score	0.92	0.26		0.59	0.88	
Support	62,310	3,789	66,099	66,099	66,099	

In Table 9, XGBoost model with SMOTE has an overall accuracy of 86% and ROC-AUC 0.76. Class 0 (non-vulnerable) achieved an accuracy score of 0.96 as well as an F1 score of 0.92. Class 1 (vulnerable) increased from previous models achieving 0.18 accuracy and 0.43 recall, which provides an average F1 score of 0.26. Susceptible samples are easy to identify after applying SMOTE and this increases class balance but tuning of Class 1 precision and recall is still required.

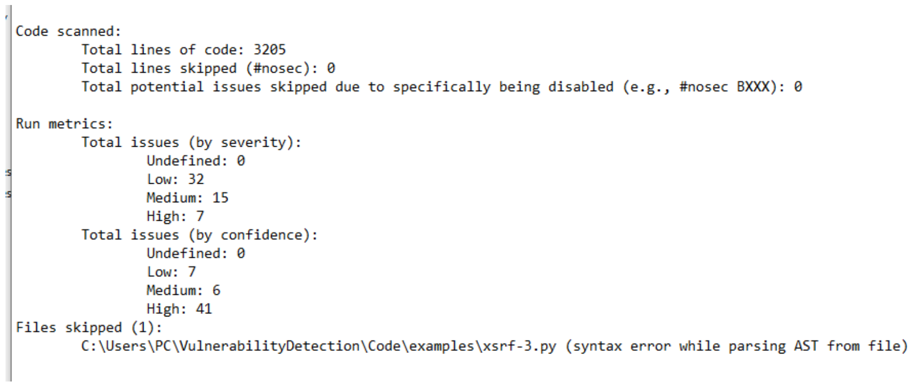
Table 10: LightGBM with SMOTE Results

Metric	Class 0 (non-vulnerable)	Class 1 (vulnerable)	Accuracy	Macro Avg	Weighted Avg	ROC-AUC Score
Precision	0.97	0.18	0.84	0.57	0.92	0.77
Recall	0.86	0.49		0.67	0.84	
F1-Score	0.91	0.26		0.59	0.87	
Support	62,310	3,789	66,099	66,099	66,099	

In Table 10, the model also includes LightGBM with SMOTE having an 84% accuracy and 0.77 ROC AUC. Non-Vulnerable or Class 0 achieved an accuracy of 0.97 and an F1 Score of 0.91. The vulnerable class 1 improved with a recall of 0.49 while the F1-score was 0.26 and accuracy remained low at 0.18. Compared to the previous model, SMOTE helped in enhancing the detection of vulnerabilities with a slight imbalance in Class 1 accuracy and F1-score. This model has a somewhat better balance than previous ensemble methods.

6.9 Analysis using SAST

(1) Bandit:



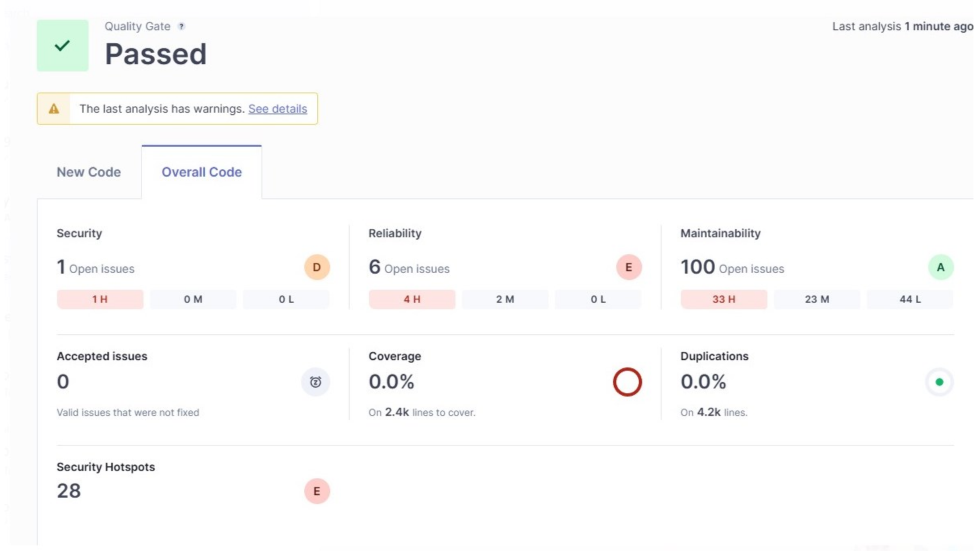
```
Code scanned:
        Total lines of code: 3205
        Total lines skipped (#nosec): 0
        Total potential issues skipped due to specifically being disabled (e.g., #nosec BXXX): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 32
                Medium: 15
                High: 7
        Total issues (by confidence):
                Undefined: 0
                Low: 7
                Medium: 6
                High: 41
Files skipped (1):
        C:\Users\PC\VulnerabilityDetection\Code\examples\xsrf-3.py (syntax error while parsing AST from file)
```

Figure 6: Output of Bandit

Figure 6 are the output results from the VUDENC dataset. There were no lines skipped because of #nosec comments and based on the Bandit scan data, the scan was completed on 3205 lines of code. The study reports 54 problems, out of which 32 are of low priority, 15 of medium priority, and 7 of high priority. Of these, 41 issues have high confidence, and the rest of them, have low- or medium-confidence. Uniquely, there was a syntax error during AST processing and one file which is xsrf-3.py was excluded; this file requires close look or coding correction. To enhance the reliability as well as the security of the obtained code, therefore, the study underscores the importance of addressing high-severity and high-confidence issues. This should be done perhaps when reviewing stalled or inactive files for possible threats.

(2) SonarQube:


Quality Gate
Passed
Last analysis 1 minute ago
The last analysis has warnings. See details
New Code | Overall Code

Security
1 Open issues D
1 H | 0 M | 0 L

Reliability
6 Open issues E
4 H | 2 M | 0 L

Maintainability
100 Open issues A
33 H | 23 M | 44 L

Accepted issues
0
Valid issues that were not fixed

Coverage
0.0%
On 2.4k lines to cover.

Duplications
0.0%
On 4.2k lines.

Security Hotspots
28 E

Figure 7: Output of SonarQube

Figure 7 are the results from the VUDENC dataset. Based on the SonarQube it is found that a few areas need improvement, however, the quality gate was cleared. There is a risk that must be fixed first because there is still one that possesses a high problem (severe) that was reported in the security section. There are 6 unresolved problems in the code base structure 4 are critical and 2 are medium even though they reside in the reliability section. This has in turn the risk of impacting the reliability of the code. Of the 100 outstanding issues related to maintainability, 33 hours is estimated to be the time required for the worst 100 issues. The other 100 looks like they need further research as out of the 24,000 lines checked, some lines remain untested, and there is no code coverage. In total, there are 28 manual security review required code fragments that must be scanned to assess the threat they might cause.

6.10 Limitations

Ensemble learning was not performed on VUDENC because of issues with the dataset split provided in the repository, which did not have proper training and validation sets and was not clearly mentioned in their respective repository too. SAST tools were not applicable to DiverseVul since the dataset contained incomplete code snippets, which these tools cannot analyze. Ensemble learning was applied to DiverseVul, while Bandit and SonarQube were used on VUDENC, giving their respective suitability for static and dynamic analysis.

Pre-processing for both methods in these two datasets was not practical due to time and resource limitations. Hence based on both dataset's compatibility, the decision was made to use these methods on different datasets. The comparison demonstrated the strength of each approach in their contexts respectively rather than a direct competition.

7 Conclusion and Future Work

7.1 Conclusion

Based on this research, it is established that both ensemble learning algorithms and SAST tools detect software defects. Comparing Random Forest with other ensemble techniques while using the evaluation metrics Accuracy, precision for the vulnerable class, and Recall for the 'non-vulnerable' class, it was observed that the Random Forest's results were the best. Including SMOTE enhanced the model in detecting weakness of the minority class since it has a problem of class imbalance. The traditional SAST technology primarily targets rule-based vulnerability discovery on code and may not offer a perfect prediction and adaptation to complex patterns and may have higher numbers of false positives. Random Forest was more effective for the precise determination of vulnerability since it can work with different data sets and reduce false negatives. Since SAST techniques and ensemble learning models are synergistic, this research suggests implementing these techniques in parallel to detect and eliminate software flaws.

7.2 Future Work

This work can be extended in the future to enhance the integration of SAST tools with ensemble learning algorithms to offer an ensembled vulnerability detection option. To deal with class imbalance, advanced preprocessing methods such as dynamic data enhancement might be applied. Transformers used for categorizing the vulnerability that uses deep learning as its base may increase the rates of correct detection and minimize false negatives. The inclusion of datasets from another programming language will greatly enhance the study's applicability. Ultimately, running the proposed models online with real limitations explored can specify the models' scalability and robustness.

References

Baptista, T., Oliveira, N. and Henriques, P.R., 2021. Using machine learning for vulnerability detection and classification. In *10th Symposium on Languages, Applications and Technologies (SLATE 2021)*.

Bhardwaj, P., 2022. *Finding IoT privacy issues through malware Detection using XGBoost machine learning technique* (Doctoral dissertation, Dublin, National College of Ireland).

Chehab, M., 2020. *Knowledge Discovery Data (KDD).* [Online] Available at: https://medium.com/analytics-vidhya/knowledge-discovery-data-kdd-a8b41509bff9

Galinde, D.R., 2023. *Effective approach for Malware Detection using Machine Learning and Deep Learning for IoT-Devices* (Doctoral dissertation, Dublin, National College of Ireland).

Harer, J.A., Kim, L.Y., Russell, R.L., Ozdemir, O., Kosta, L.R., Rangamani, A., Hamilton, L.H., Centeno, G.I., Key, J.R., Ellingwood, P.M. and Antelman, E., 2018. Automated software vulnerability detection with machine learning. *arXiv preprint arXiv:1803.04497*.

Kim, D.K. and Chung, Y.K., 2024. Addressing class imbalances in software defect detection. *Journal of Computer Information Systems*, *64*(2), pp.219-231.

LauraWartschinski, 2019. *VUDENC - Vulnerability Detection with Deep Learning on a Natural Codebase.* [Online] Available at: https://github.com/LauraWartschinski/VulnerabilityDetection/tree/master?tab=readme-ov-file

Li, K., Chen, S., Fan, L., Feng, R., Liu, H., Liu, C., Liu, Y. and Chen, Y., 2023, November. Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 921-933).

Li, K., Xue, Y., Chen, S., Liu, H., Sun, K., Hu, M., Wang, H., Liu, Y. and Chen, Y., 2024. Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We? *Proceedings of the ACM on Software Engineering*, *1*(FSE), pp.1447-1470.

Li, Z., Zou, D., Tang, J., Zhang, Z., Sun, M. and Jin, H., 2019. A comparative study of deep learning-based vulnerability detection system. *IEEE Access*, *7*, pp.103184-103197.

Lin, G., Wen, S., Han, Q.L., Zhang, J. and Xiang, Y., 2020. Software vulnerability detection using deep neural networks: a survey. *Proceedings of the IEEE*, *108*(10), pp.1825-1848. 24

Logan, T., 2020. A practical, iterative framework for secondary data analysis in educational research. *The Australian educational researcher*, *47*(1), pp.129-148.

Pang, Y., Xue, X. and Namin, A.S., 2016, December. Early identification of vulnerable software components via ensemble learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 476-481). IEEE.

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Vats, I., Moazen, H. and Sarro, F., 2021. A survey on machine learning techniques for source code analysis. *arXiv preprint arXiv:2110.09610*.

Siewruk, G. and Mazurczyk, W., 2021. Context-aware software vulnerability classification using machine learning. *IEEE Access*, *9*, pp.88852-88867.

surrealyz, 2024. *DiverseVul: A New Vulnerable Source Code Dataset for Deep Learning Based Vulnerability Detection.* [Online] Available at: https://github.com/wagner-group/diversevul

Wang, L., Li, X., Wang, R., Xin, Y., Gao, M. and Chen, Y., 2020. PreNNsem: A heterogeneous ensemble learning framework for vulnerability detection in software. *Applied Sciences*, *10*(22), p.7954.

Zaharia, S., Rebedea, T. and Trausan-Matu, S., 2022. Machine Learning-Based Security Pattern Recognition Techniques for Code Developers. *Applied Sciences*, *12*(23), p.12463.