Empowering SW Security: CodeBERT and Machine Learning Approaches to Vulnerability Detection

Lov Kumar, Vikram Singh, Srivalli Patel and Pratyush Mishra

National Institute of Technology, Kurukshetra

Abstract

Software (SW) systems experience faults after deployment, raising concerns about reliability and leading to financial losses, reputational damage, and safety risks. This paper presents a novel approach using CodeBERT, a state-of-the-art neural code representation model pre-trained in multi-programming languages and employs various code metrics to predict SW faults. The study comprehensively evaluates trained models by analyzing publicly available codebase and employing diverse machine learning (ML) models, feature selection techniques (FSTs), and class balancing through Synthetic Minority Oversampling Technique (SMOTE). The results show that SMOTE significantly enhances vulnerability detection performance, particularly in accuracy, AUC, sensitivity, and specificity. The EXTR classifier consistently outperforms others, with an average AUC of 0.82, and the features selected using the genetic algorithm (GA) feature selection technique, despite achieving a mean AUC of 0.84. Interestingly, among employed embedding techniques, SW metrics combined with CodeBERT (SMCBERT) stand out as top performers, achieving the highest mean AUC score of 0.80, making models trained on SMCBERT the best for SW vulnerability prediction.

1 Introduction

Software(SW) vulnerability refers to a flaw arising from SW design, development, or configuration errors that can be exploited to breach explicit or implicit security policies (Ghaffarian and Shahriari, 2017). SW vulnerability is a particular defect that creates security risks and poses a significant concern for SW development as it is often used to launch attacks. Vulnerabilities, ranging from coding errors to design flaws, create entry points for attackers to exploit systems, steal data, disrupt services, and cause significant financial or reputational damage to businesses and governments. So, the vulnerability detection process is critical mainly due to the pervasive nature of SW in modern society and the increasing sophistication of cyber threats. Effective vulnerability detection helps identify and remediate these weaknesses before they can be exploited, reducing the likelihood and impact of the attacks.

Recent research in the realm of SW security indicates that vulnerability detection is essential in promoting the reliable development of SW and efficiently allows developers to reduce the number of vulnerabilities patched after the production phase (Hanif et al., 2021). In most studies, SW vulnerability detection is a data-driven method in SW quality assurance, leveraging historical information on vulnerabilities to predict if specific parts of code (functions, files, or code snippets) are susceptible to security weaknesses. Contemporary methodologies such as the utilization of code reviews can be valuable but also expensive and time-consuming for developers, requiring them to consider potential attack scenarios and solutions (Mäntylä and Lassenius, 2008). Additionally, reviewing every file for vulnerabilities might be impractical for massive code repositories.

In this context, the utilization of ML offers significant advantages for vulnerability detection by enabling the analysis of large code bases at scale and identifying known and unknown threats through pattern recognition. Unlike traditional methods that rely on predefined rules, ML models can adapt to emerging vulnerabilities by retraining with new data, ensuring up-to-date protection, and also excel at detecting complex, multi-layered vulnerabilities that might elude manual analysis. This reduces human error while automating the detection process, making ML a more dynamic, scaleable, and accurate approach to securing SW environments.

In this paper, we have comprehensively investigated the performance of several commonly used ML and DL techniques for SW vulnerability detection on a codebase extracted from publicly avail-

able datasets, e.g. Drupal, Moodle, and PHPMyAdmin. In this context, the CodeBert source code embedding technique is utilized to capture code semantics into comprehensive vectors that may be sufficiently used by the pipeline, after which a set of diverse FSTs were employed for the identification of the most informative features, effectively trimming the data, to enhance ML models in several ways including improved accuracy, faster training times, and greater interpretability. Furthermore, a class balancing technique, SMOTE, is applied due to the inherent imbalance in datasets. By combining six vectorization techniques, 15 classification algorithms, 05 FSTs, and a class balancing technique on the proposed datasets, we aim to gain insights into how these methods may contribute towards more effective vulnerability detection pipelines.

2 Related Work

The existing research efforts on SW vulnerabilities fall under two broad categories: traditional ML and deep learning (DL) approaches. Both categories make use of various data preprocessing techniques, often relying on traditional metrics and tests. The potential research work is briefly discussed below:

2.1 Traditional ML-based models

Chernis (2018) (Chernis and Verma, 2018) presented a methodology for analyzing features extracted from C source code. They defined functions to extract both trivial and non-trivial features. Finally, they considered various classifiers, such as Naive Bayes, k-nearest neighbors, and Random Forests to classify the test samples. Medeiros (2020) (Medeiros et al., 2020) conducted a comprehensive experiment to assess the effectiveness of SW metrics using ML algorithms. The study extracted vulnerability-related insights from SW metrics of notable C/C++ projects such as Mozilla Firefox, Linux Kernel, Apache HTTPd, Xen, and Glibc. The main finding is that ML algorithms, combined with SW metrics, can identify vulnerable code units with high confidence in security-critical systems. However, this approach is less effective for low-critical or non-critical systems due to a high number of false positives.

Pereiraetal.(Pereira et al., 2021) used static analysis tool (SAT) alerts and SW metrics (SMs) as inputs for ML algorithms to predict vulnerabilities. Data from the Mozilla project, supplemented with static analysis alerts from CPP check and Flawfinder, are used. Results show that models using SMs outperform those using SAT alerts alone, but neither approach achieves satisfactory precision and recall simultaneously. Notably, Napier et al. (Napier et al., 2023) evaluated the text-based ML model's effectiveness using seven ML models, five natural language processing techniques, and three data processing methods are employed in experiments. Results indicate that condensed functions with fewer features often yield better prediction results within the same model, but overall, text-based ML models struggle to detect vulnerabilities consistently across different projects and types.

Similarly, Bilgin et al. (Bilgin et al., 2020) introduced an ML-based vulnerability prediction method and present a technique for vectorial source code representation. The experimental analysis demonstrated the utility of partial Abstract Syntax Tree (AST) representations for vulnerability prediction. Jabeenetal. (Jabeen et al., 2022) empirically studied various ML techniques and statistical methods to predict SW vulnerabilities across different datasets. The ML techniques include cascade-forward and feed-forward backpropagation neural networks, ANFIS, multi-layer perceptron, SVM, and bagging. Statistical methods like the Alhazmi-Malaiya model, linear regression, and logistic regression are also evaluated. Results show that ML techniques significantly outperform statistical models.

Munonye and Péter (Munonye and Péter, 2022) presented an ML-based approach for detecting potential vulnerabilities in the OAuth authentication and authorization flow. The study treated the OAuth protocol as a supervised learning problem, developing, tuning, and evaluating seven classification models. Exploratory Data Analytics (EDA) techniques were utilized to extract and analyze specific OAuth features. The developed models underwent training, tuning, and testing, resulting in a performance accuracy exceeding 90% for vulnerability detection in the OAuth authentication and authorization flow. When compared with known vulnerabilities, the model achieves a match rate of 54%.

2.2 Deep learning-based prediction models

Chakraborty et al.(Chakraborty et al., 2022) utilized the FFMPeg+Qemu dataset proposed by Zhou et al. to assess the technique's performance in

real-world settings. The authors curated a new vulnerability dataset from two large-scale popular projects (Chromium and Debian). Directly employing pre-trained models for real-world vulnerability detection results in an average performance decrease of approximately 73% , and retraining these models with real-world data, their performance dips by roughly 54% compared to the reported results. Similarly, Zhuang et al. (Zhuang et al., 2021) introduced 3GNN. 3GNN improved vulnerability detection by analyzing source code graph components (AST, CFG, DFG) separately, using CGCN and SAG Pooling for initial vector representations. These representations were refined through hierarchical modules and merged into an MLP for classification. Evaluations on C/C++ datasets (Draper, QEMU+FFmpeg) show that 3GNN outperforms baselines, achieving a 6.9%.

Ban et al. (Ban et al., 2019) proposed a performance evaluation of SW vulnerability detection based on deep-learn features. The author used deep learning algorithms (i.e., BiLSTM) to extract features. Even if it is possible to show results using ML techniques, they suffer poor performance on cross-project and class imbalance problems in SW vulnerability detection.

Similarly, Tang et al. (Tang et al., 2020) conducted experiments to test the performance of Bi-LSTM and RVFL on SW vulnerability detection problems using data preprocessing methods (i.e., the vector representation and the program symbolization methods) and found that RVFL trains faster, but Bi-LSTM is more accurate. Doc2vec enhances training speed and generalization over word2vec, while multi-level symbolization improves model precision.

Liu et al. (Liu et al., 2020) proposed a DL-based method using BiLSTM for AST representation that tackles cross-project feature extraction and class imbalance in vulnerability detection. It employs fuzzy-based oversampling (FOS) to generate synthetic vulnerable samples, rebalancing the dataset and improving the identification of vulnerable programming patterns. Wartschinski et al. (Wartschinski et al., 2022) introduced VUDENC (Vulnerability Detection with Deep Learning on a Natural Codebase), a deep learning-based vulnerability detection system for automatically learning features of vulnerable code from a large, real-world codebase. Trained on a dataset of security fixes, VUDENC achieves 78%–87% recall and 82%-96% precision, outperforming competitors.

The proposed framework offers key advancements over traditional approaches. While ML models using software metrics or static analysis tools often rely on hand-crafted features and struggle with high false positive rates and class imbalances, the hybrid approach combining CodeBERT embeddings with software metrics provides a more effective feature representation. This enhances the capture of both semantic and structural aspects of code, improving model robustness and adaptability across different software projects as compared to using single-dimensional embeddings or metrics..

Deep learning models, though promising for vulnerability detection, often face performance issues like overfitting, cross-project inconsistencies, and class imbalance in real-world datasets. The use of SMOTE and feature selection in this framework addresses these, improving generalization and accuracy. Additionally, ML-based methods are more flexible for real-time applications, being less resource-intensive and deployable in environments with limited computational power, making them ideal for industries with stringent resource constraints, as is the general case in vulnerability-centric frameworks.

3 Proposed SW Vulnerability Prediction framework

Figure 1 illustrates an overview of the proposed research study, which aims to design a range of SW vulnerability prediction pipelines in conjunction with different data preprocessing techniques and ML/ DL techniques. The investigation begins with a public dataset, leveraging data from three distinct SW projects. The first step of the proposed solution is to extract semantics from the source code. So, we have applied three approaches to represent source code as a vector: SW metrics and two embedding like TF-IDF and CodeBert. These vectors are applied as input, both individually and combined, to the SW vulnerability prediction models. Furthermore, four distinct FSTs are applied to the extracted features, resulting in a total of 75 datasets (5 sets of features * 3 projects * (All features +4 FSTs)).

Next, our objective is to ensure a balanced representation within each dataset. A unique data balancing technique is integrated with each of the 75 datasets obtained from the feature selection phase, effectively doubling the data. Finally, to compre-

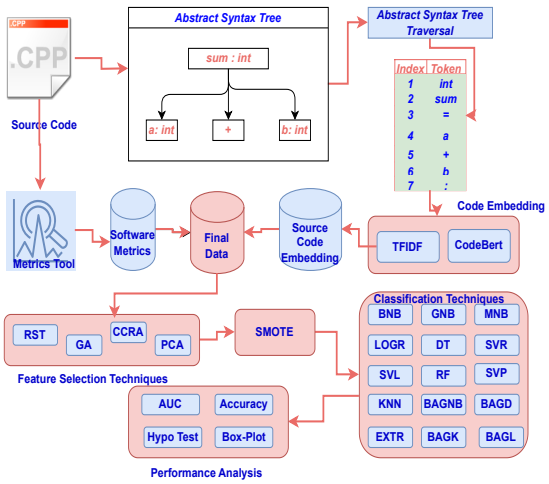


Figure 1: Schematic view of proposed Vulnerability Detection framework

hensively assess the predictive power of various algorithms, we evaluate the performance of 15 classification techniques on each of the 150 culminated datasets. This comprehensive evaluation yields valuable insights into the most effective classification pipelines for vulnerability detection tasks. The details of each participating entity, highlighted through Figure 1, are provided below:

3.1 Dataset Details

The study conducted in this paper adapts three public datasets: Drupal, Moodle and PHPMyAdmin (Riccardo Scandariato, 2008), in which data is stored in the form of text mining and SW metrics, with two columns in the text mining datasets. The first column is the token column, which contains individual lexical tokens derived from the code snippets. The second column 'lab' consists of binary values, (0 or 1) 0 representing non-vulnerability and 1 for vulnerability; hence, the computational task is that of binary classification. For Moodle, the data source was the National Vulnerability Database (NVD) (of Standards and Technology, 2008), while for Drupal and PHPMyAdmin, we used the security reports managed by those projects. Applications can be vulnerable to different types of vulnerabilities. The semantics of each adapted dataset is as follows:

- **Drupal**, a widely used content management system, consists of 202 files, 62 of which are labelled as vulnerable. The most prevalent type of vulnerabilities are authorization vulnerabilities, due to it's design as a multi-user application since initialization, leading to more opportunities for improper authorization implementation.

- **Moodle** is an open-source learning management system with 2,942 files, 24 of which are marked as vulnerable. The predominant vulnerabilities are related to authorization, similar to Drupal, due to its multi-user nature.

- **PHPMyAdmin** is a web-based management tool for the MySQL database. There are 322 total files, of which 27 are marked vulnerable. In contrast, the cominant vulnerability in this dataset is cross-site scripting (XSS) vulnerabilities, as the dataset is focused less on user interactions compared to Moodle and Drupal.

3.2 Embedding Techniques

After data collection, we mainly aim to convert the source code into vectorized notations and make it a suitable format for easy classification. To find the vectorized notations, we used the pre-trained model CodeBERT, a multi-programming-lingual model pre-trained on NL-PL, i.e. Natural language and Programming language, pairs in six programming languages (Python, Java, JavaScript, PHP, Ruby, Go). We have also used TF-IDF to find the vectorized notations and source code metrics to measure quantifiable or countable SW characteristics. So, in this study, 05 sets of input features, namely SW metrics (SOFT), TFIDF, CBERT, SW metrics with TF-IDF (SMFIDF), and SW metrics with SMCBERT, are utilized for predictions.

3.3 Feature Extraction Techniques

To reduce data complexity and improve the efficiency and performance of ML algorithms, we employ a total of 05 FSTs, including AF (the consideration of all features as important), specifically RST, CCRA, PCA, and GA.

3.4 Data Balancing Techniques

This study uses SMOTE to generate synthetic samples for the minority class. This process increases the number of minority-class samples, aiming to create a more balanced dataset.

3.5 Classification Techniques

Finally, we have applied different classification models, namely Multinomial Naive Bayes (MNB),

Table 1: Accuracy and AUC values for SW vulnerability Prediction models

	AUC								Accuracy							
	BNB	**KNN**	**LOGR**	**DT**	**SVCL**	**BAGKN**	**RF**	**EXTR**	**BNB**	**KNN**	**LOGR**	**DT**	**SVCL**	**BAGKN**	**RF**	**EXTR**
ORG Data																
All Features																
SOFT	0.51	0.61	0.79	0.75	0.77	0.79	0.77	0.77	68.81	71.78	77.72	71.29	74.26	72.28	76.24	75.74
TFIDF	0.79	0.68	0.80	0.64	0.77	0.75	0.83	0.83	70.79	70.30	73.27	71.78	70.79	70.30	79.21	78.71
CBERT	0.69	0.60	0.79	0.79	0.74	0.79	0.77	0.78	54.46	71.78	80.69	74.75	78.22	75.74	77.23	74.26
SMFIDF	0.81	0.69	0.81	0.72	0.74	0.78	0.83	0.81	74.75	76.24	79.21	76.73	74.26	72.77	77.23	79.70
SMCBRT	0.70	0.65	0.81	0.80	0.75	0.79	0.76	0.78	54.95	78.22	77.23	77.23	72.28	77.23	74.26	73.27
GA																
SOFT	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	69.31	70.79	75.25	77.23	73.76	78.71	73.27	77.23
TFIDF	0.77	0.71	0.80	0.77	0.72	0.77	0.82	0.81	93.56	100.00	100.00	95.54	100.00	93.56	88.61	99.01
CBERT	0.88	1.00	1.00	0.90	1.00	0.88	0.96	0.97	58.42	100.00	100.00	88.61	100.00	86.14	74.75	89.11
SMFIDF	0.69	0.67	0.84	0.77	0.80	0.77	0.86	0.85	90.10	100.00	100.00	97.52	100.00	95.54	86.14	93.56
SMCBRT	0.92	1.00	1.00	0.94	1.00	0.95	0.92	0.95	59.90	100.00	99.51	88.61	100.00	83.66	83.17	88.12
SMOTE																
All Features																
SOFT	0.53	0.78	0.78	0.84	0.78	0.83	0.87	0.87	52.14	75.36	70.00	77.86	72.14	75.71	80.36	81.79
TFIDF	0.91	0.83	0.92	0.88	0.91	0.88	0.93	0.93	79.64	79.64	80.00	78.93	80.36	78.57	80.71	84.64
CBERT	0.65	0.77	0.91	0.86	0.88	0.84	0.88	0.90	63.57	73.93	81.07	74.64	82.50	74.64	79.64	81.07
SMFIDF	0.91	0.79	0.93	0.87	0.93	0.89	0.92	0.94	83.57	80.71	85.00	80.00	85.71	77.50	85.36	82.14
SMCBRT	0.72	0.79	0.93	0.89	0.91	0.88	0.92	0.95	62.50	76.07	81.43	74.29	83.21	73.21	84.29	80.71
GA																
SOFT	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	54.64	80.00	71.07	78.57	73.57	76.79	79.64	80.36
TFIDF	0.88	0.76	0.88	0.84	0.88	0.85	0.92	0.91	87.14	100.00	100.00	99.64	100.00	96.79	93.57	97.86
CBERT	0.91	1.00	1.00	0.97	1.00	0.92	0.97	0.99	64.29	100.00	100.00	86.07	100.00	79.64	91.79	97.50
SMFIDF	0.89	0.79	0.91	0.86	0.91	0.89	0.92	0.93	81.79	100.00	100.00	98.21	100.00	98.21	96.07	98.57
SMCBRT	0.90	1.00	1.00	0.97	1.00	0.98	0.95	0.98	65.71	100.00	100.00	87.14	100.00	90.00	93.57	92.50

Bernoulli Naive Bayes (BNB), Gaussian Naive Bayes (GNB), Decision Trees (DT), K Nearest Neighbors (KNN), SVM with linear (SVML), polynomial (SVMP), RBF kernel (RBFK), Bagging Classifier with base estimator as K Neighbors Classifier, MultinomialNB, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, ExtraTrees Classifier, AdaBoost Classifier, and Gradient Boosting Classifier for SW vulnerability detection. MLP classifiers with optimization algorithms blogs, Adam and SD, have also been adapted for SW vulnerability detection. These models are validated using 5-fold cross-validations, and the effectiveness of developed prediction models is assessed using a variety of performance metrics, including accuracy and area under the ROC curve. We further utilized box plots for visual representation and the Wilcoxon rank-sum test to detect significant differences among the models.

4 Experimental Results

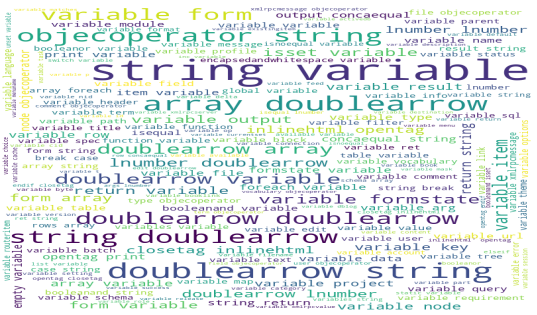
The overall predictive ability assessment of the proposed research framework is shown in Figure 1, described in this section. Table 1 lists the predictive ability of the selected combination of word embedding, data sampling, feature selection, and classification techniques on public datasets. The results for other combinations are of similar type. The models are validated using 5-fold cross-validation (CV). The key inferences from Table 1 are as follows:

- High AUC scores confirm that the developed prediction models correctly predict the source code containing SW vulnerability.
- Prediction models trained on embedding with source code metrics have a high predictive ability. In contrast, those models trained on an individual, either embedding or source code metrics, have low predictive ability.
- Models trained on SMOTE balanced data are better than those trained on the original dataset (ORGD).
- Models trained after feature selection using GA have a very high predictive ability.

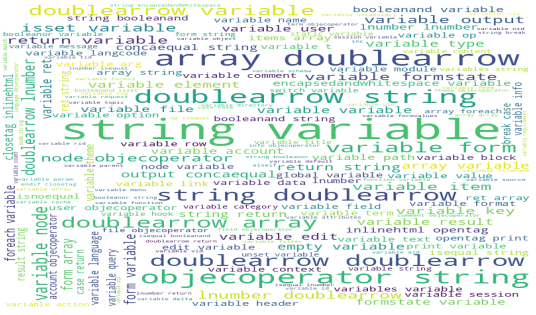
4.1 Comparative evaluation of Embedding techniques

Understanding Contextual Cues using Word Clouds: Figure 2(a) and Figure 2(b) illustrate word-cloud of both non-vulnerable and vulnerable source code, respectively. These visuals reveal the context in which vulnerabilities arise, highlight key features, functions, variables, or words that dominate the non-vulnerable data, and establish key

differences in programming practices or patterns between the two datasets.



((a)) Non-Vulnerability



((b)) Vulnerability

Figure 2: Word-level contextual overlapping among Vulnerable and Non-vulnerable source code datasets

Comparative analysis of Code Embedding Techniques Performance through Box-Plot Analysis: Figure 3 illustrates a detailed analysis of the performance differences of sets of features for prediction. Upon investigation, it is evident that SMCBERT stands out as a top performer, achieving the highest mean AUC score of 0.80. CBERT alone closely trails behind its soft-metrics-enhanced counterpart, with a median AUC of 0.78. TF-IDF exhibits the same AUC value, indicating the importance of textual frequency across the word embedding landscape for the datasets at hand. It is interesting to observe that while the combination of soft metrics and CBERT causes a boost in accuracy, the same combination with TFIDF achieves marginally lower scores, stressing the importance of careful integration with soft metrics and the additional context it provides. The simple use of SW metrics alone delivers the lowest degree of accurate performance, highlighting the importance of developing cohesive combination techniques.

Exploring Code Embedding Techniques through Friedman Mean Rank: We have used the Friedman Mean Rank test with a confidence interval of 95% (0.05 significance level), meaning the null hypothesis is accepted if p≥0.05 to find

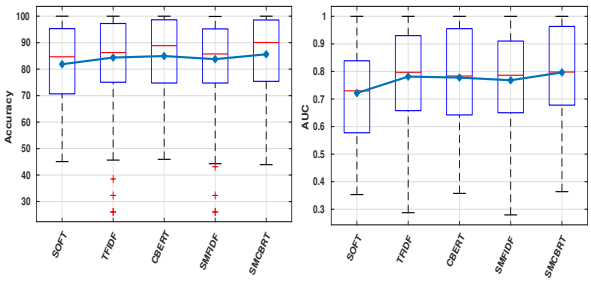


Figure 3: Performance box-plots of Code-Embedding techniques

the significant impact of the feature sets extracted from the source code. The obtained p-value after applying the Friedman Mean Rank test is less than 0.05, confirming that the model's prediction power depends on the methods used to find the features from the source code. A model with a lower mean rank value performs better than the one with a higher mean rank value. The models trained on a combination of SW metrics with CodeBERT(SMCBERT) lower mean rank, as shown in Table 2, represent the best models for SW vulnerability prediction. The results of Table 2 also confirm that the models trained on an individual, either embedding or source code metrics, have low performance for SW vulnerability prediction.

Table 2: Friedman Mean Rank statistics of Code-Embedding Techniques at p-value 0.05

	SOFT	TFIDF	CBERT	SMFIDF	SMCBRT
Accuracy	3.39	2.86	3.01	2.94	2.79
AUC	3.78	2.77	3.11	2.77	2.57

4.2 Evaluation of Feature Selection Methodologies

We employed four diverse FSTs to identify and retain the most informative characteristics of the data while discarding irrelevant ones, ultimately leading to simpler and potentially more effective prediction models. We compared the original feature set with four additional sets generated by the selection procedures.

Comparative analysis of FST performance through Box-Plots: Figure 4 illustrates assessment metrics accuracy and AUC across the original dataset and four feature selection methods. An important observation drawn from this analysis is that CCRA shows marginally higher performance than AF, with an 85% accuracy score, with RST only marginally behind with a mean of 84%. For com-

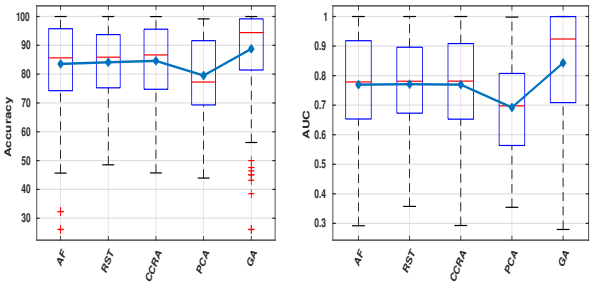


Figure 4: Performance Box-plots of FS Techniques

parison, AF results in a mean accuracy score of 83%, and in terms of AUC, all the aforementioned features along with AF show identical scores of 0.77. This highlights that certain features make up for most of the interpretation of the dataset in the context of the vulnerability prediction task at hand, and the use of FSTs allows for a reduction in the complexity of the data without compromising overall performance. It is apparent that the GA model displays the highest mean accuracy at 88% as well as mean AUC at 0.84 and the smallest interquartile range (IQR). This may highlight the superior predictive capabilities of the GA model compared to other FSTs. In contrast, PCA shows the lowest degree of accurate performance, significantly lower than AF. This suggests that dimensionality reduction primarily based on variance might not align with the features most relevant to the fault prediction task.

Exploring FST Techniques through Friedman Mean Rank: We have applied the Friedman Mean Rank test with a confidence interval of 95% (0.05 significance level), meaning the null hypothesis is accepted if p≥0.05 to find the significant impact of the features used as an input of the models. The Friedman Mean Rank test return p-value <=0.05 confirms that the models trained by changing the input features significantly impact the prediction power of the models. From Table 3, we can conclude that the models trained using selected sets of features derived by GA as input give the best results and perform significantly better than all features. This finding confirms that the models' performance will not be degraded after removing irrelevant features.

4.3 Feasibility analysis of Class balancing

Optimizing data distribution in classes is crucial for classification model performance. This study uses the data-balancing technique SMOTE to address the essential problem of the underrepresented minority class, which may be prevalent in SW vulnerability prediction tasks.

Table 3: Friedman Mean Rank of FST, at p-value 0.05

	AF	RST	CCRA	PCA	GA
Accuracy	2.95	3.01	2.92	3.78	2.35
AUC	2.67	2.92	2.92	4.24	2.25

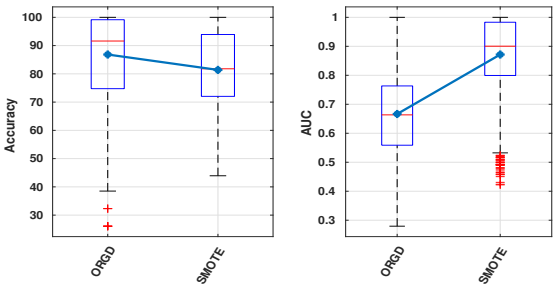


Figure 5: Performance Box-plots of Class Balancing techniques

Comparative analysis of Data Balancing Techniques performance through Box-Plot: Figure 5 showcases the evaluation metrics for both the original datasets and the datasets that have undergone balancing techniques. In terms of accuracy, the mean value for the original data is 86%, whereas the balanced data exhibits a slightly lower mean accuracy of 81%. This suggests a potential decrease in accuracy when utilizing data balancing techniques. However, a different trend emerges when we consider the AUC metric. The original data boasts an AUC of 0.67, while the balanced data demonstrates a significantly higher AUC of 0.87. This substantial improvement in AUC compared to the marginal difference in accuracy leads to the conclusion that the model can differentiate between classes when using balanced data.

Exploring Data Balancing Techniques through Friedman Mean Rank: This work also uses the Friedman Mean Rank test with a confidence interval of 95% (0.05 significance level), meaning the null hypothesis is accepted if p≥0.05 to find the significant impact of the models trained on balanced data using SMOTE. The calculated p-value <=0.05 of the Friedman Mean Rank test confirms that the models trained on balanced data significantly impact the performance of the models. A model with a lower mean rank value performs better than the one with a higher mean rank value. Hence, from Table 4, we can conclude that the models trained on balanced data give the best results and perform sig-

nificantly better than the original data. The above finding is based on the AUC value because of the imbalanced nature of the data.

Table 4: Friedman Mean Rank of Data Balancing Techniques, at p-value 0.05

	ORGD	**SMOTE**
Accuracy	1.45	1.55
AUC	1.94	1.06

4.4 Evaluation of Classification techniques

We analyzed the performance of nine different ML classifiers and six ensemble Techniques based on accuracy performance metrics. Our primary objective was to gain insights into the effectiveness of these classifiers in the predictive model.

Exploring ML techniques performance through Box-Plot: Figure 6 reveals a range of mean accuracy values from 75% to 90% , indicating significant variations in predictive capabilities among classifiers. In the ML-based classification, vulnerability detection models demonstrate a slight variation in performance is noticed when the focus is shifted from accuracy score to AUC score and vice versa. However, it is clear that the EXTR Classifier demonstrated the highest mean accuracy and AUC score at 90.87% and 0.82, respectively, followed by LOGR and RF classifiers - yielding values only marginally lower. Consequently, the consistently higher performance of ensemble classifiers like Extra Trees highlights their superior ability to manage the intricate patterns and interactions inherent in the dataset.

Exploring ML techniques through Friedman Mean Rank: This work also uses the Friedman Mean Rank test with a confidence interval of 95% (0.05 significance level), meaning the null hypothesis is accepted if p$\geq$0.05 to find the significant impact of the models. The calculated p-value <=0.05 of the Friedman Mean Rank test confirms that the models trained by using different ML algorithms significantly impact the performance of the models. From Table 5, we can conclude that the models trained using EXTR classifiers give the best results and perform significantly better than other classifiers.

5 Conclusion

The role of code embedding techniques in designing a reliable and accurate SW vulnerability detection model, alongside various ML/DL techniques, feature selection, and data balancing methods, remains an open and hypothetical area of research. In this context, we systematically evaluated different classifiers and ensemble methods, alongside advanced embedding techniques like CodeBERT. Interestingly, the feasibility of embedding techniques, SOFT, TFIDF, CBERT, SMFIDF, and SMCBERT is explored for the intended task. Upon investigation, it is evident that SW metrics combined with CodeBERT (SMCBERT) stand out as top performers, achieving the highest mean AUC score of 0.80, making models trained on SMCBERT the best for SW vulnerability prediction. The proposed framework delivers 150 vulnerability prediction pipelines and is evaluated on publicly available datasets from Drupal, Moodle, and PHPMyAdmin. The experimental findings show that FSTs, particularly GA, significantly enhance model performance by identifying the most informative features. Data balancing methods like SMOTE further improve model generalization, as seen in higher AUC scores. Additionally, among classifiers, EXTR demonstrates a superior ability to manage the intricate patterns and interactions in the vulnerability datasets.

The future scope of this work includes exploring advanced FSTs, more sophisticated data balancing and adaptive learning methods could enhance model generalization. Expanding to diverse, modern datasets will ensure scalability, while real-time detection and cross-platform evaluations can demonstrate effectiveness in dynamic environments. The approach could also automate vulnerability detection within continuous integration pipelines, providing developers with timely feedback on code security.

References

Xinbo Ban, Shigang Liu, Chao Chen, and Caslon Chua. 2019. A performance evaluation of deep-learnt features for software vulnerability detection. *Concurrency and Computation: Practice and Experience*, 31(19):e5103. E5103 cpe.5103.

Zeki Bilgin, Mehmet Akif Ersoy, Elif Ustundag Soykan, Emrah Tomur, Pinar Çomak, and Leyli Karaçay. 2020. Vulnerability prediction from source code using machine learning. *IEEE Access*, 8:150672–150684.

Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9):3280–3296.

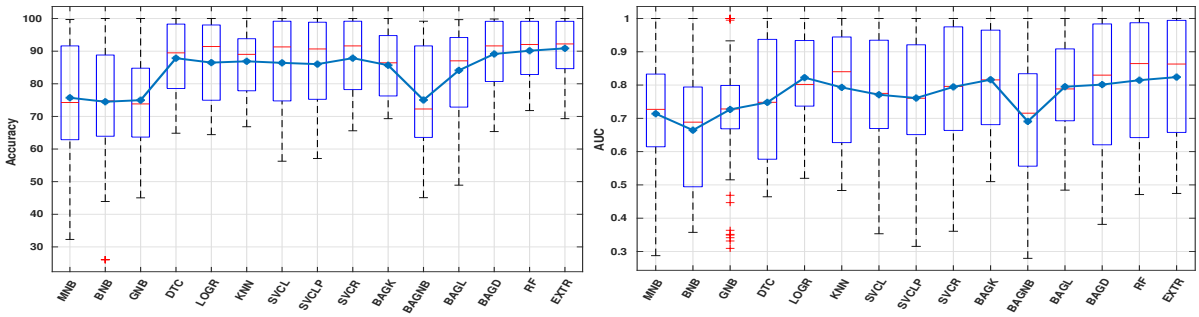


Figure 6: Performance Box-Plots of Classification Techniques

Table 5: Friedman Mean Rankm of ML classifiers, at p-value 0.05

	MNB	BNB	GNB	DTC	LOGR	KNN	SVCL	SVCLP	SVCR	BAGK	BAGNB	BAGL	BAGD	RF	EXTR
Accuracy	10.98	11.70	12.13	8.72	6.48	7.40	6.74	7.66	5.44	7.63	11.21	8.36	5.96	5.05	4.53
AUC	10.17	11.44	10.21	10.12	5.35	7.92	7.99	8.35	6.27	6.38	11.77	7.79	6.54	5.33	4.36

Boris Chernis and Rakesh Verma. 2018. Machine learning methods for software vulnerability detection. In *Proceedings of the fourth ACM international workshop on security and privacy analytics*, pages 31–39.

Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM computing surveys (CSUR)*, 50(4):1–36.

Hazim Hanif, Mohd Hairul Nizam Md Nasir, Mohd Faizal Ab Razak, Ahmad Firdaus, and Nor Badrul Anuar. 2021. The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, 179:103009.

Gul Jabeen, Sabit Rahim, Wasif Afzal, Dawar Khan, Aftab Ahmed Khan, Zahid Hussain, and Tehmina Bibi. 2022. Machine learning techniques for software vulnerability prediction: a comparative study. *Applied Intelligence*, 52(15):17614–17635.

Shigang Liu, Guanjun Lin, Qing-Long Han, Sheng Wen, Jun Zhang, and Yang Xiang. 2020. Deepbalance: Deep-learning and fuzzy oversampling for vulnerability detection. *IEEE Transactions on Fuzzy Systems*, 28(7):1329–1343.

Mika V Mäntylä and Casper Lassenius. 2008. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):430–448.

Nadia Medeiros, Naghmeh Ivaki, Pedro Costa, and Marco Vieira. 2020. Vulnerable code detection using software metrics and machine learning. *IEEE Access*, 8:219174–219198.

Kindson Munonye and Martinek Péter. 2022. Machine learning approach to vulnerability detection in oauth 2.0 authentication and authorization flow. *International Journal of Information Security*, 21(2):223–237.

Kollin Napier, Tanmay Bhowmik, and Shaowei Wang. 2023. An empirical study of text-based machine learning models for vulnerability detection. *Empirical Software Engineering*, 28(2):38.

National Institute of Standards and U.S. Department of Commerce Technology. 2008. National vulnerability dataset. https://nvd.nist.gov/. [Online; accessed 19-July-2008].

José D'Abruzzo Pereira, Joao R Campos, and Marco Vieira. 2021. Machine learning to combine static analysis alerts with software metrics to detect security vulnerabilities: An empirical study. In *2021 17th European Dependable Computing Conference (EDCC)*, pages 1–8. IEEE.

or James Walden Riccardo Scandariato, Jeffrey Stuckman. 2008. Dataset replication repository. https://seam.cs.umd.edu/webvuldata/studydata.html. [Online; accessed 19-July-2008].

Gaigai Tang, Lianxiao Meng, Huiqiang Wang, Shuangyin Ren, Qiang Wang, Lin Yang, and Weipeng Cao. 2020. A comparative study of neural network techniques for automatic software vulnerability detection. In *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 1–8.

Laura Wartschinski, Yannic Noller, Thomas Vogel, Timo Kehrer, and Lars Grunske. 2022. Vudenc: vulnerability detection with deep learning on a natural codebase for python. *Information and Software Technology*, 144:106809.

Yufan Zhuang, Sahil Suneja, Veronika Thost, Giacomo Domeniconi, Alessandro Morari, and Jim Laredo. 2021. Software vulnerability detection via deep learning over disaggregated code graph representation.