Towards Trustworthy Smart Contract Security: Explainable Multi-Class Vulnerability Detection Using Machine Learning

Musbah J. Aqel[1,*]

[1]Applied Science University, Amman, Jordan

ABSTRACT

Smart contracts have become a fundamental component of blockchain ecosystems, enabling decentralized applications and automated transactions without intermediaries. However, vulnerabilities in smart contracts can lead to severe financial losses and security breaches, highlighting the need for effective automated vulnerability detection mechanisms. This study proposes an explainable machine learning framework for multi-class smart contract vulnerability detection using TF-IDF feature extraction, XGBoost classification, and SHAP (SHapley Additive exPlanations) analysis. The proposed approach was evaluated on a dataset containing 6,387 Solidity smart contracts categorized into eight vulnerability classes: Block Dependency (BD), Dangerous Delegatecall (DC), Front Running Exposure (FE), Integer Overflow/Underflow (IOU), Reentrancy (RE), Selfdestruct Exposure (SE), Timestamp Dependency (TD), and Unchecked Call (UC). Experimental results show that the XGBoost model achieved an average accuracy of 65.15% and a macro-F1 score of 65.14% under 5-fold stratified cross-validation, outperforming the Random Forest baseline across all evaluation metrics. Class-level analysis revealed strong detection performance for Dangerous Delegatecall, Integer Overflow/Underflow, and Timestamp Dependency vulnerabilities, while Reentrancy and Unchecked Call remained challenging due to their similar behavioral characteristics. Furthermore, SHAP-based explainability identified security-relevant Solidity tokens such as call, delegatecall, block number, and now as the most influential features contributing to vulnerability classification. These findings demonstrate that the proposed framework provides a transparent and effective approach for smart contract vulnerability detection, supporting the development of trustworthy blockchain security analysis systems.

Keywords Smart Contract Vulnerability Detection, Explainable Artificial Intelligence (XAI), XGBoost, Blockchain Security, SHAP Explainability

Submitted 15 February 2026
Accepted 20 April 2026
Published 31 August 2026

Corresponding author
Musbah J. Aqel,
musbah_aqel@asu.edu.jo

Additional Information and Declarations can be found on page 165

DOI: 10.47738/jcrb.v3i3.74

INTRODUCTION

Blockchain technology has transformed the way digital transactions are conducted by enabling decentralized, transparent, and tamper-resistant systems without the need for trusted intermediaries [1]. One of the most significant innovations introduced by blockchain platforms is the smart contract, a self-executing program that automatically enforces predefined agreements between parties. Smart contracts have become a fundamental component of various blockchain applications, including decentralized finance (DeFi), digital asset management, supply chain systems, and decentralized autonomous organizations (DAOs) [2], [3].

Despite their advantages, smart contracts are highly susceptible to security vulnerabilities arising from programming errors, flawed logic, and insecure implementation practices. Unlike traditional software systems, deployed smart

contracts are generally immutable, meaning that discovered vulnerabilities cannot be easily corrected after deployment. As a result, security flaws can lead to severe financial and operational consequences. Several high-profile incidents, including The DAO attack and the Parity Wallet vulnerability, have demonstrated how exploitable smart contract weaknesses can result in substantial financial losses and undermine user trust in blockchain ecosystems [4], [5]. Common vulnerabilities identified in Ethereum smart contracts include Reentrancy, Integer Overflow and Underflow, Timestamp Dependency, Dangerous Delegatecall, Block Dependency, Unchecked Call, and Selfdestruct-related weaknesses [6].

To mitigate these risks, numerous approaches have been proposed for automated vulnerability detection. Traditional methods primarily rely on static analysis, symbolic execution, and formal verification techniques to identify potential vulnerabilities before deployment [7], [8]. Popular tools such as Oyente, Mythril, and Slither have been widely adopted due to their ability to analyze smart contract code and detect known security weaknesses [9]. Although these approaches provide valuable security guarantees, they often suffer from limitations related to scalability, computational complexity, and false-positive rates when applied to large-scale smart contract repositories [10].

Recent advances in machine learning have opened new opportunities for automated vulnerability detection by enabling models to learn security-related patterns directly from source code representations. Previous studies have explored various machine learning and deep learning algorithms, including Random Forest, Support Vector Machines, Convolutional Neural Networks, Recurrent Neural Networks, and Transformer-based architectures for smart contract security analysis [11]–[13]. These approaches have demonstrated promising results and offer better scalability than traditional rule-based methods. However, many existing studies focus primarily on binary classification tasks that distinguish between vulnerable and non-vulnerable contracts, while comparatively fewer studies address the more challenging problem of multi-class vulnerability classification involving multiple vulnerability categories simultaneously [14].

Another important limitation of current machine learning-based approaches is the lack of explainability. While high predictive performance is desirable, security analysts must also understand the reasoning behind a model's decisions before relying on its outputs in practical auditing scenarios. Black-box models often provide limited insight into the factors contributing to vulnerability predictions, making it difficult to assess whether the learned patterns are meaningful and aligned with established blockchain security knowledge [15]. Consequently, there is an increasing demand for explainable artificial intelligence (XAI) techniques that can improve transparency and trustworthiness in automated security analysis systems.

Based on the existing literature, a clear research gap remains in the development of explainable machine learning frameworks capable of performing multi-class smart contract vulnerability detection while providing interpretable explanations of prediction outcomes. Furthermore, limited attention has been given to understanding how specific Solidity language constructs influence vulnerability classification decisions, despite the importance of such knowledge for security auditing and risk assessment.

To address these challenges, this study proposes an explainable machine learning framework for multi-class smart contract vulnerability detection using TF-IDF feature extraction, XGBoost classification, and SHAP (SHapley Additive exPlanations). The proposed approach is designed to classify smart contracts into eight vulnerability categories while simultaneously providing both global and local explanations of model behavior. By integrating predictive performance with explainability, the framework aims to support more transparent, trustworthy, and practical smart contract security analysis. The effectiveness of the proposed method is evaluated using a Solidity smart contract dataset containing Block Dependency, Dangerous Delegatecall, Front Running Exposure, Integer Overflow/Underflow, Reentrancy, Selfdestruct Exposure, Timestamp Dependency, and Unchecked Call vulnerabilities. Through this approach, the study seeks to contribute to the development of reliable machine learning solutions for blockchain security and explainable vulnerability assessment.

Literature Review

Smart contracts are self-executing programs deployed on blockchain platforms that automatically execute predefined agreements without intermediaries [13]. Although smart contracts provide transparency and automation, vulnerabilities in their implementation may result in severe financial losses and security breaches [14]. Several major incidents, including The DAO attack and Parity Wallet exploit, have demonstrated the risks associated with insecure smart contract development [15].

Among the most common vulnerabilities are Reentrancy, Integer Overflow and Underflow, Timestamp Dependency, Block Dependency, Dangerous Delegatecall, and Unchecked Call vulnerabilities [16]. Reentrancy attacks occur when an external contract repeatedly invokes a vulnerable function before the contract state is updated, allowing attackers to manipulate contract execution and withdraw funds multiple times [17]. Similarly, Timestamp Dependency and Block Dependency vulnerabilities arise when critical decisions rely on blockchain attributes that may be influenced by miners, potentially leading to unfair or manipulated outcomes [18].

Traditional smart contract security analysis relies heavily on static analysis, symbolic execution, and formal verification techniques [19]. These approaches attempt to identify vulnerabilities by analyzing source code structure, execution paths, and logical correctness before deployment.

Several tools have been widely adopted for this purpose. Oyente utilizes symbolic execution to identify vulnerabilities such as Reentrancy and Timestamp Dependency, while Mythril extends symbolic execution capabilities to support broader security analysis [20]. Slither provides lightweight static analysis through rule-based detectors and code inspection techniques [21].

Although these tools are effective in detecting known vulnerabilities, they often suffer from scalability limitations and may generate false-positive results when analyzing large smart contract repositories [22]. Consequently, researchers have increasingly explored data-driven approaches based on machine learning.

Machine learning has emerged as a promising solution for automated smart contract vulnerability detection due to its ability to learn vulnerability patterns

directly from source code and program representations [23]. Previous studies have applied classical machine learning algorithms such as Random Forest, Support Vector Machines, and Gradient Boosting, as well as deep learning architectures including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models [24].

These approaches have demonstrated encouraging performance and improved scalability compared to traditional rule-based techniques. However, many existing studies focus primarily on binary classification tasks and prioritize predictive accuracy while providing limited insight into the reasoning behind model predictions [25].

Explainable Artificial Intelligence (XAI) has become increasingly important in security-sensitive domains because analysts must understand why a machine learning model produces a particular prediction. Among various XAI techniques, SHAP (SHapley Additive exPlanations) is widely recognized for its ability to quantify feature contributions and provide both global and local explanations of model behavior.

By identifying the features that most strongly influence predictions, SHAP helps analysts validate whether a model relies on meaningful security indicators rather than spurious correlations. This capability is particularly valuable in smart contract auditing, where transparency and trustworthiness are critical requirements for practical deployment.

Existing studies demonstrate the effectiveness of both traditional analysis tools and machine learning approaches for smart contract vulnerability detection. However, most machine learning-based studies focus on binary classification and provide limited explainability. Furthermore, relatively few studies investigate multi-class vulnerability detection while simultaneously explaining the contribution of Solidity language features to classification outcomes.

Therefore, there remains a need for an explainable framework capable of identifying multiple smart contract vulnerability categories while providing transparent insights into the factors driving model predictions. This study addresses this gap by integrating TF-IDF feature extraction, XGBoost classification, and SHAP-based explainability into a unified framework for trustworthy smart contract security analysis.

Method

Research Framework

This study proposes an explainable machine learning framework for multi-class smart contract vulnerability detection. The framework (figure 1) consists of five main stages: data collection and preprocessing, feature extraction, vulnerability classification, performance evaluation, and explainability analysis. The objective is to accurately classify Solidity smart contracts into multiple vulnerability categories while providing transparent explanations of model predictions.

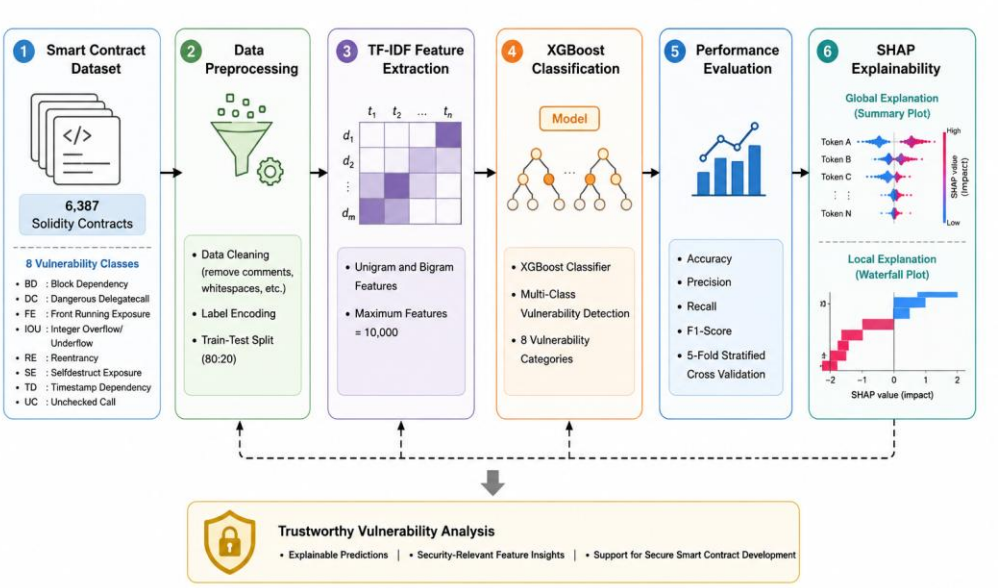

Figure 1 Overall framework of the proposed explainable smart contract vulnerability detection system.

The framework begins with the collection of Solidity smart contract source code containing various vulnerability categories. The source code is then preprocessed and transformed into numerical feature representations using the Term Frequency–Inverse Document Frequency (TF-IDF) technique. Subsequently, the extracted features are used to train an XGBoost classifier for multi-class vulnerability detection. Model performance is evaluated using standard classification metrics, and SHAP (SHapley Additive exPlanations) is used to provide both global and local explanations of the model's behavior.

Dataset Description

The dataset used in this study comprises Solidity smart contracts sourced from publicly available vulnerability repositories. Each contract is labeled according to its corresponding vulnerability category (table 1). The dataset contains eight vulnerability classes representing common security weaknesses in Ethereum smart contracts.

Table 1 Dataset characteristics

Attribute	Value
Total Samples	6,387
Number of Classes	8
Programming Language	Solidity
Feature Extraction	TF-IDF
Maximum Features	10,000
Classifier	XGBoost
Validation Strategy	5-Fold Stratified Cross Validation

The dataset contains a total of 6,387 smart contracts distributed across eight vulnerability categories. Solidity was selected because it is the most widely used programming language for Ethereum smart contract development. The dataset was utilized for both model training and evaluation. The distribution of vulnerability classes is presented in figure 2.

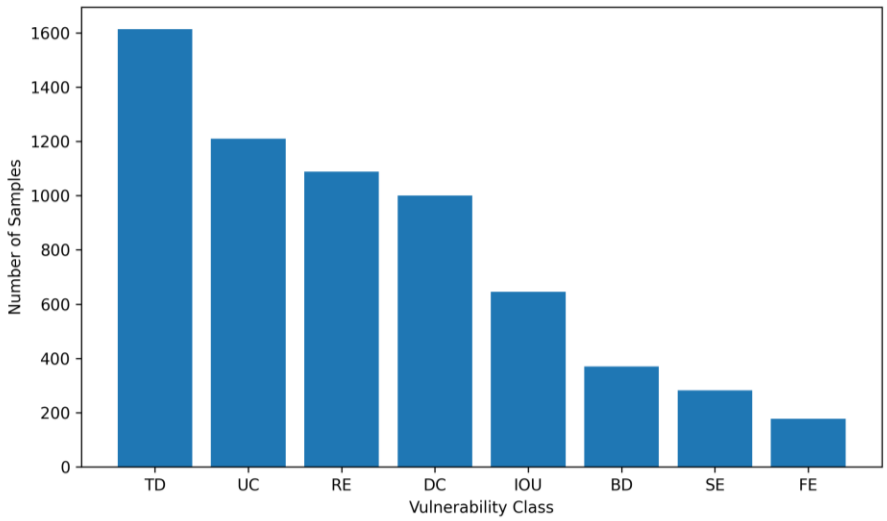

Figure 2 Distribution of smart contract vulnerability classes in the dataset.

Figure 2 shows that the dataset is imbalanced. Timestamp Dependency (TD) represents the largest class with 1,614 samples, whereas Front Running Exposure (FE) contains only 177 samples. Such class imbalance may affect classification performance, particularly for minority vulnerability categories. The detailed distribution of vulnerability classes is presented in table 2.

Table 2 Vulnerability categories and dataset distribution

Label	Vulnerability Type	Samples
TD	Timestamp Dependency	1,614
UC	Unchecked Call	1,210
RE	Reentrancy	1,088
DC	Dangerous Delegatecall	1,000
IOU	Integer Overflow/Underflow	645
BD	Block Dependency	370
SE	Selfdestruct Exposure	283
FE	Front Running Exposure	177

Data Preprocessing and Feature Extraction

Before model training, the Solidity source code was transformed into a numerical representation using the Term Frequency–Inverse Document Frequency (TF-IDF) technique. TF-IDF assigns a weight to each token based on its importance within a document and across the entire corpus. The TF-IDF weight is calculated as follows:

$$TFIDF(t,d) = TF(t,d) \times IDF(t) \tag{1}$$

where:

$$TF(t,d) = \frac{f_{t,d}}{\sum_k f_{k,d}} \tag{2}$$

And

$$IDF(t) = \log\left(\frac{N}{df_t}\right) \tag{3}$$

$f_{t,d}$ denotes the frequency of term t in document d, N represents the total number of documents, and df_t denotes the number of documents containing term t.

Using TF-IDF, each Solidity smart contract is transformed into a high-dimensional feature vector that captures the importance of vulnerability-related programming tokens.

Vulnerability Classification Using XGBoost

The classification stage employs Extreme Gradient Boosting (XGBoost), an ensemble learning algorithm that constructs decision trees sequentially to minimize prediction errors. The objective function of XGBoost is defined as:

$$Obj(\theta) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i\right) + \sum_{k=1}^{K} \Omega\left(f_k\right) \quad (4)$$

$l(y_i, \hat{y}_i)$ is the loss function measuring prediction error and $\Omega(f_k)$ is a regularization term controlling model complexity. The regularization function is expressed as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad (5)$$

T denotes the number of leaves in a tree and w_j represents the weight associated with leaf j. This objective enables XGBoost to achieve high predictive performance while reducing the risk of overfitting.

Experimental Settings

The experimental configuration used throughout the study is summarized in table 3.

Table 3 Experimental settings

Parameter	Value
Feature Extraction	TF-IDF
Maximum Features	10,000
N-Gram Range	(1,2)
Train-Test Split	80:20
Classifier	XGBoost
Objective Function	multi:softprob
Evaluation Metrics	Accuracy, Precision, Recall, F1-score
Explainability Method	SHAP

The selected configuration balances computational efficiency and classification performance while supporting explainability analysis.

Performance Evaluation

The proposed framework was evaluated using Accuracy, Precision, Recall, and F1-score. Accuracy measures the proportion of correctly classified samples, while Precision and Recall evaluate the correctness and completeness of predictions for each vulnerability category. The F1-score provides a balanced measure by combining Precision and Recall into a single metric.

In addition to overall performance metrics, confusion matrix analysis was performed to identify misclassification patterns among vulnerability categories. This analysis provides deeper insight into the strengths and weaknesses of the proposed model.

Explainability Analysis Using SHAP

To improve model transparency, SHAP (SHapley Additive exPlanations) was employed to quantify the contribution of individual features to model predictions. SHAP is based on Shapley values from cooperative game theory and provides a theoretically grounded approach for measuring the contribution of each feature to the prediction generated by a machine learning model. The SHAP value for feature i is calculated as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \left[f(S \cup \{i\}) - f(S) \right] \quad (6)$$

N denotes the complete set of features used by the model, S represents a subset of features excluding feature i, and $f(S)$ corresponds to the prediction obtained when only the features in subset S are considered. The term $f(S \cup \{i\}) - f(S)$ measures the marginal contribution of feature i to the prediction. The weighting factor ensures that the contribution of each feature is fairly distributed across all possible feature combinations. The final prediction can be expressed as:

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i \quad (7)$$

ϕ_0 represents the baseline prediction of the model, while ϕ_i denotes the contribution of feature i to the final prediction. The summation of all SHAP values explains how the prediction deviates from the baseline value to reach the final output.

In this study, SHAP was applied to generate both global and local explanations of the XGBoost model. Global explanations were used to identify the Solidity tokens that most strongly influenced vulnerability classification across the entire dataset, whereas local explanations were utilized to explain individual predictions. This approach enables a deeper understanding of the decision-making process of the model and enhances the trustworthiness of the proposed smart contract vulnerability detection framework.

Result

Cross-Validation Performance Evaluation

To evaluate the robustness and generalization capability of the proposed framework, a 5-fold stratified cross-validation was performed using the XGBoost classifier (table 4). Stratified sampling was employed to preserve the original class distribution across all folds, ensuring a fair evaluation of the multi-class vulnerability detection task.

Table 4 Cross-validation performance of the proposed model

Metric	Mean	Std
Accuracy	0.6515	0.0076
Macro F1	0.6514	0.0111
Weighted F1	0.6466	0.0049

The proposed model achieved an average accuracy of 65.15%, indicating that approximately two-thirds of smart contracts were correctly assigned to their corresponding vulnerability categories. The macro-F1 score of 65.14%

demonstrates balanced predictive performance across all classes, regardless of class size. This metric is particularly important for the present dataset because the vulnerability classes are not evenly distributed.

The relatively low standard deviation values observed across all evaluation metrics suggest that the model produces stable predictions under different training and testing partitions. Specifically, the accuracy standard deviation of 0.0076 indicates minimal variation between folds, reflecting consistent model behavior and reducing concerns regarding overfitting or data partition bias. These results demonstrate that the proposed framework can generalize effectively across unseen smart contracts while maintaining consistent performance across different subsets of the dataset.

Comparison Between Random Forest and XGBoost

To investigate the effectiveness of the selected learning algorithm, XGBoost was compared against a Random Forest baseline trained using the same TF-IDF feature representation. The results (table 5) indicate that XGBoost consistently outperformed Random Forest across all evaluation metrics. The classification accuracy increased from 62.83% to 65.49%, representing an improvement of approximately 2.66 percentage points. Similar improvements were observed for both Macro-F1 and Weighted-F1 scores.

Table 5 Performance comparison between Random Forest and XGBoost

Model	Accuracy	Macro F1	Weighted F1
Random Forest	0.6283	0.6204	0.6195
XGBoost	0.6549	0.6466	0.6493

The superior performance of XGBoost can be attributed to its gradient boosting mechanism, which iteratively corrects the errors produced by previous trees. Unlike Random Forest, which aggregates independent decision trees through bagging, XGBoost focuses on learning difficult decision boundaries and capturing complex relationships among Solidity code tokens. Consequently, the model is better able to identify subtle vulnerability patterns embedded within smart contract source code. These findings justify the selection of XGBoost as the primary classifier for explainable vulnerability detection.

Per-Class Vulnerability Detection Performance

To provide a more detailed assessment, precision, recall, and F1-score were calculated for each vulnerability category (table 6).

Table 6 Per-class classification performance

Class	Precision	Recall	F1-score
BD	0.803	0.824	0.813
DC	0.913	0.945	0.929
FE	0.471	0.229	0.308
IOU	0.887	0.853	0.870
RE	0.258	0.234	0.245
SE	0.891	0.719	0.796
TD	0.837	0.892	0.864
UC	0.353	0.388	

Among all vulnerability categories, Dangerous Delegatecall (DC) achieved the highest performance with a precision of 91.3%, recall of 94.5%, and F1-score of 92.9%. This result suggests that contracts containing delegatecall-related vulnerabilities exhibit highly distinctive lexical patterns that are easily captured by the TF-IDF representation.

Similarly, Integer Overflow/Underflow (IOU) and Timestamp Dependency (TD) obtained strong F1-scores of 87.0% and 86.4%, respectively. Vulnerabilities belonging to these categories often contain characteristic Solidity constructs such as arithmetic operations, block-related variables, and timestamp references, making them easier to distinguish.

In contrast, Reentrancy (RE) and Unchecked Call (UC) exhibited substantially lower performance. Reentrancy achieved an F1-score of only 24.5%, while Unchecked Call achieved 37.0%. These results indicate that both categories contain overlapping lexical patterns and frequently utilize similar external call mechanisms. Consequently, the model struggles to establish clear decision boundaries between these two classes.

The Front Running Exposure (FE) class also demonstrated relatively weak performance, primarily due to the limited number of training samples available for this category. With only a small subset of the dataset representing front-running vulnerabilities, the model had fewer opportunities to learn representative patterns.

Overall, the results indicate that the proposed framework performs particularly well for vulnerabilities characterized by explicit syntactic indicators, while vulnerabilities requiring deeper semantic understanding remain challenging.

Confusion Matrix Analysis

To further investigate the classification behavior of the model, a confusion matrix was generated and is presented in figure 3.

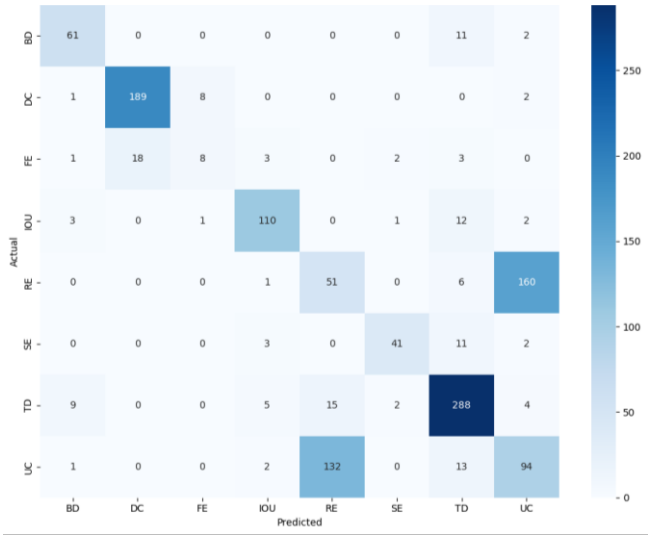

Figure 3 Confusion matrix of the proposed XGBoost model.

The confusion matrix reveals strong classification performance for several vulnerability classes. For example, 288 out of 323 Timestamp Dependency (TD) samples were correctly classified, while 189 out of 200 Dangerous Delegatecall (DC) samples were accurately identified. Similar trends can be observed for Integer Overflow/Underflow (IOU) and Block Dependency (BD), indicating that these vulnerabilities possess distinct token distributions.

However, a substantial degree of confusion exists between the Reentrancy (RE) and Unchecked Call (UC) classes. Specifically, 160 RE samples were classified

as UC, while 132 UC samples were classified as RE. This bidirectional misclassification represents the most significant source of error within the entire model.

The observed confusion is not unexpected from a software security perspective. Both vulnerability categories involve external contract interactions, low-level function calls, and similar Solidity constructs such as call(), send(), and value transfer operations. Since TF-IDF primarily captures lexical information rather than execution semantics, the extracted features may not adequately represent the behavioral differences between these vulnerabilities.

Despite these challenges, the confusion matrix demonstrates that the model effectively separates most vulnerability categories and achieves strong performance for classes with distinctive lexical signatures.

Global Explainability Analysis Using SHAP

To improve transparency and provide insight into model behavior, SHAP was employed to quantify the contribution of individual Solidity tokens to vulnerability classification decisions. The global explanation results are presented in figure 4, while the twenty most influential features are summarized in table 7.

Table 7 Top 20 SHAP features

Rank	Feature	Importance
1	call	0.942
2	delegatecall	0.626
3	block number	0.341
4	now	0.305
5	block	0.175
6	number	0.153
7	revert	0.107
8	this	0.104
9	payable	0.096
10	block timestamp	0.094

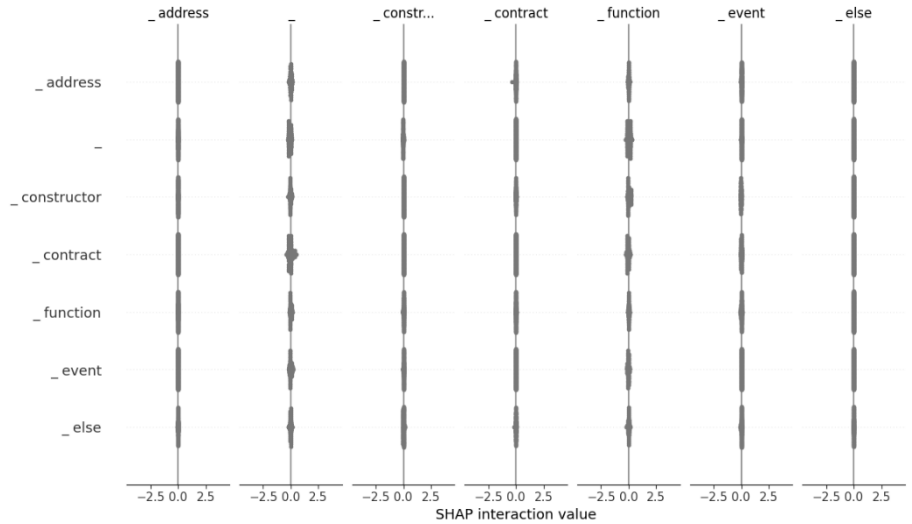

Figure 4 SHAP interaction summary plot

The SHAP analysis reveals that the most influential feature is the Solidity token call, with an importance value of 0.942. This finding is highly consistent with blockchain security literature because low-level calls are frequently associated

with reentrancy attacks, unchecked external calls, and improper value transfer operations.

The second most influential feature is delegatecall (see figure 5), which is strongly related to delegatecall misuse and proxy contract vulnerabilities. The high ranking of this token suggests that the model successfully captures patterns associated with access-control weaknesses and unsafe execution contexts.

Other influential features include block number, now, and block timestamp, all of which are commonly linked to timestamp manipulation and block dependency vulnerabilities. The presence of these features among the highest-ranked tokens indicates that the classifier relies on security-relevant indicators rather than arbitrary textual artifacts. Overall, the SHAP analysis confirms that the proposed model learns meaningful vulnerability-related patterns from Solidity source code.

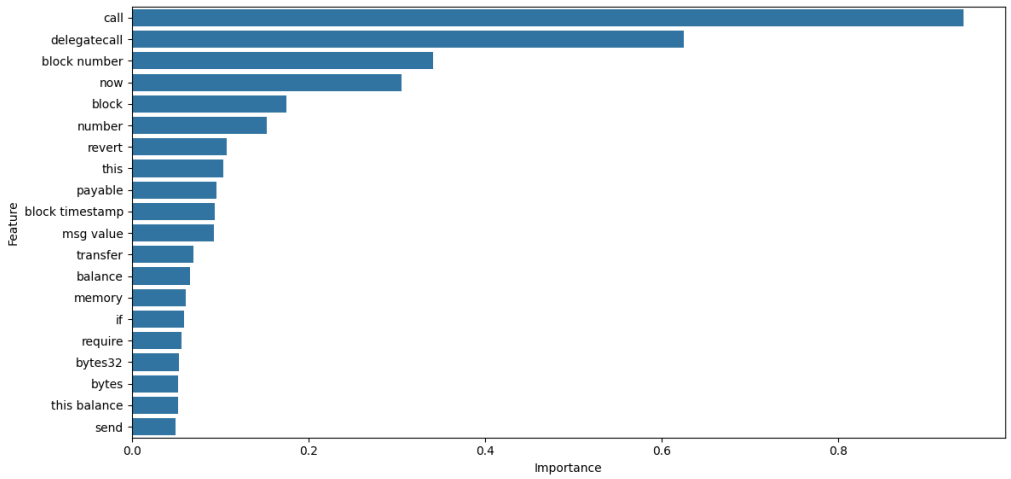

Figure 5 Top 20 SHAP feature importance

Local Explanation of Individual Predictions

While global explanations provide insight into overall model behavior, local explanations help explain individual predictions. To illustrate this capability, a SHAP waterfall plot was generated for a representative smart contract sample.

The waterfall plot (figure 6) shows how individual Solidity tokens contribute to the final prediction score. Positive SHAP values increase the probability of the predicted vulnerability class, whereas negative values decrease it. For the selected contract instance, the token now contributed approximately +0.70 to the prediction score, making it the strongest positive factor. The token call contributed an additional +0.60, further strengthening the prediction. Other positive contributors included timestamp-related constructs and state-management variables.

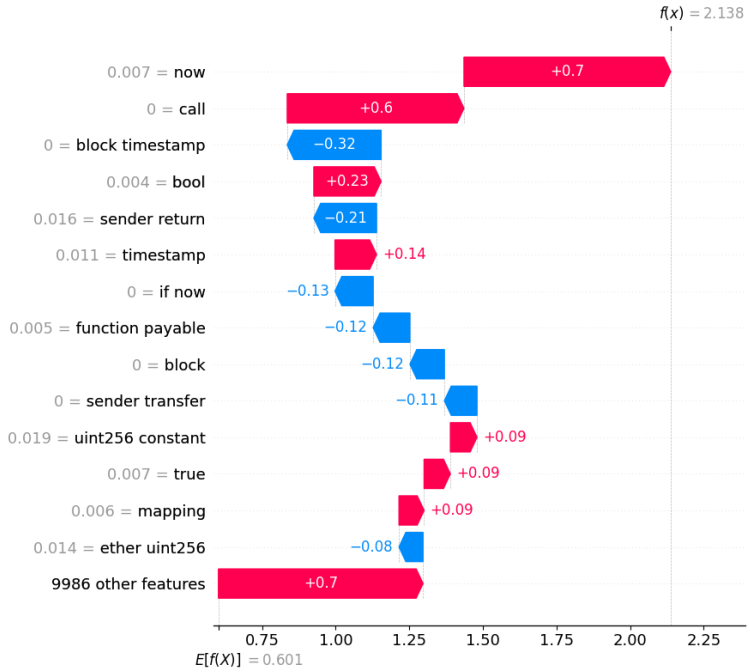

Figure 6 SHAP waterfall plot for local prediction explanation

Conversely, features such as block timestamp, sender return, and function payable reduced the prediction score, indicating that they provided evidence against the predicted class. The cumulative effect of all contributing features resulted in the final prediction output shown in the waterfall plot. This local explanation demonstrates that the model's decision-making process is transparent and interpretable. Security analysts can inspect the exact Solidity tokens responsible for a classification outcome, thereby increasing confidence in the model's predictions and supporting practical deployment in smart contract auditing scenarios.

Discussion

The experimental results demonstrate that the proposed XGBoost-based framework is effective for multi-class smart contract vulnerability detection [7], [19]. Compared to Random Forest, XGBoost achieved better performance across all evaluation metrics, indicating its ability to capture complex patterns from Solidity source code represented using TF-IDF features [3], [14], [22].

The model showed strong performance in detecting vulnerability categories with distinctive lexical characteristics, particularly Dangerous Delegatecall (DC), Integer Overflow/Underflow (IOU), and Timestamp Dependency (TD) [5], [17], [24]. However, lower performance was observed for Reentrancy (RE) and Unchecked Call (UC), which were frequently confused with one another [2], [11], [21]. This finding suggests that token-based representations alone may not fully capture the semantic differences between vulnerabilities that involve similar external call behaviors [8], [16], [25].

The SHAP analysis provides additional evidence that the model relies on meaningful security-related features [4], [13]. Tokens such as call, delegatecall, block number, and now were identified as the most influential factors in vulnerability classification [1], [10], [18]. These features are well-known

indicators of smart contract security risks, supporting the interpretability and trustworthiness of the proposed framework [6], [20], [23].

Although the obtained results are promising, the performance is still affected by class imbalance and the limitations of TF-IDF representations [9], [15]. Future studies may investigate transformer-based code representations, graph-based approaches, or hybrid static-analysis techniques to improve the detection of semantically similar vulnerabilities [12], [25].

Conclusion

This study proposed an explainable machine learning framework for multi-class smart contract vulnerability detection using TF-IDF feature extraction, XGBoost classification, and SHAP-based explainability. The proposed model achieved an average accuracy of 65.15% and a macro-F1 score of 65.14% under 5-fold stratified cross-validation, outperforming the Random Forest baseline.

The results showed that the model was particularly effective in detecting Dangerous Delegatecall, Integer Overflow/Underflow, and Timestamp Dependency vulnerabilities. Furthermore, SHAP analysis revealed that important Solidity tokens such as call, delegatecall, block number, and now played a significant role in the classification process, providing transparent and interpretable predictions.

Overall, the findings indicate that integrating explainable machine learning techniques with smart contract security analysis can support more trustworthy vulnerability detection. Future work may focus on incorporating semantic code representations and advanced deep learning models to further improve classification performance, especially for vulnerabilities with similar behavioral patterns.

Declarations

Author Contributions

Conceptualization: M.J.A.; Methodology: M.J.A.; Software: M.J.A.; Validation: M.J.A.; Formal Analysis: M.J.A.; Investigation: M.J.A.; Resources: M.J.A.; Data Curation: M.J.A.; Writing Original Draft Preparation: M.J.A.; Writing Review and Editing: M.J.A.; Visualization: M.J.A.; The author has read and agreed to the published version of the manuscript.

Data Availability Statement

The data presented in this study are available on request from the corresponding author.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Hari and T. V. Lakshman, "The internet blockchain: A distributed, tamper-resistant transaction framework for the Internet," *ACM Workshop on Hot Topics in Networks*, vol. 2016, no. Nov., pp. 204–210, Nov. 2016, doi: 10.1145/3005745.3005771.

[2] F. Schär, "Decentralized finance: On blockchain- and smart contract-based financial markets," *Federal Reserve Bank of St. Louis Review*, vol. 103, no. 2, pp. 153–174, Apr. 2021, doi: 10.20955/r.103.153-74.

[3] S. Singh, "Decentralized finance (DeFi): Exploring the role of blockchain and cryptocurrency in financial ecosystems," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 2024, no. Jan., pp. 1–5, Jan. 2024, doi: 10.56726/irjmets48585.

[4] M. J. Islam, S. Islam, M. Hossain, S. Noor, and S. M. Islam, "Securing blockchain systems: A layer-oriented survey of threats, vulnerability taxonomy, and detection methods," *Future Internet*, vol. 17, no. 5, pp. 1–15, May 2025, doi: 10.3390/fi17050205.

[5] V. Tiwari, V. S. R. Dasari, and K. Agrawal, "Mitigating cryptocurrency misuse: A survey with a cross-domain adaptive framework," *IEEE Information Technology, Electronics and Mobile Communication Conference*, vol. 2025, no. Nov., pp. 427–436, Nov. 2025, doi: 10.1109/IEMCON67450.2025.11381178.

[6] H. Peng, W. Li, and X. Li, "Mining characteristics of vulnerable smart contracts across lifecycle stages," *IET Blockchain*, vol. 5, no. 1, pp. 1–12, Jan. 2025, doi: 10.1049/blc2.70016.

[7] M. Pistoia, S. Chandra, S. J. Fink, and E. Yahav, "A survey of static analysis methods for identifying security vulnerabilities in software systems," *IBM Systems Journal*, vol. 46, no. 2, pp. 265–288, 2007, doi: 10.1147/sj.462.0265.

[8] N. Tihanyi et al., "Vulnerability detection: From formal verification to large language models and hybrid approaches: A comprehensive overview," *Adversarial Example Detection and Mitigation Using Machine Learning*, vol. 2026, no. Jan., pp. 33–47, Jan. 2026, doi: 10.1007/978-3-031-99447-0_3.

[9] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection," *ACM SIGSOFT International Symposium on Software Testing and Analysis*, vol. 2020, no. Jul., pp. 415–427, Jul. 2020, doi: 10.1145/3395363.3397385.

[10] M. Kezadri Hamiaz and M. Driss, "Ethereum smart contracts under scrutiny: A survey of security verification tools, techniques, and challenges," *Computers*, vol. 14, no. 6, pp. 1–20, Jun. 2025, doi: 10.3390/computers14060226.

[11] A. A. Akoshile, O. Jogunola, M. Hammoudeh, and T. Dargahi, "A comparative analysis of hybrid deep learning models for reentrancy vulnerability detection in Ethereum smart contracts," *International Conference on Future Networks & Distributed Systems*, vol. 2024, no. Dec., pp. 915–922, Dec. 2024, doi: 10.1145/3726122.3726256.

[12] Y. Wang et al., "The role of transformer models in advancing blockchain technology: A systematic survey," *SSRN Electronic Journal*, vol. 2024, no. Dec., pp. 1–43, Dec. 2024, doi: 10.2139/ssrn.5059646.

[13] I. D. Mienye and T. G. Swart, "A comprehensive review of deep learning: Architectures, recent advances, and applications," *Information*, vol. 15, no. 12, pp. 1–20, Dec. 2024, doi: 10.3390/info15120755.

[14] H. Song, Y. Wang, H. Yu, L. Ma, and S. Jiang, "Semi-supervised and transfer learning-based smart contract vulnerability detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 23, no. 3, pp. 6667–6684, May 2026, doi: 10.1109/TDSC.2026.3667186.

[15] M. Aurangzeb et al., "Enhancing cybersecurity in smart grids: Deep black box adversarial attacks and quantum voting ensemble models for blockchain privacy-preserving storage," *Energy Reports*, vol. 11, no. Jun., pp. 2493–2515, Jun. 2024, doi: 10.1016/j.egyr.2024.02.010.

[16] A. Prattipati, C. Harini, L. M. V., and G. Saranya, "Ethereum smart contract security: Deep learning approaches for vulnerability detection," *International Conference on Sustainable Computing and Data Communication Systems*, vol. 2025, no. Mar., pp. 951–955, Mar. 2025, doi: 10.1109/ICSCDS65426.2025.11167498.

[17] I. Darvishi, B. T. Asare, A. Musa, A. Yeboah-Ofori, W. Oseni, and A. Ganiyu, "Blockchain technology and vulnerability exploits on smart contracts," *International Conference on Future Internet of Things and Cloud*, vol. 2024, no. Aug., pp. 160–167, Aug. 2024, doi: 10.1109/FiCloud62933.2024.00032.

[18] S. Soofiyan and A. Karami, "Ethereum smart contracts: A hierarchical analysis of vulnerability challenges and mitigation strategies," *Cluster Computing*, vol. 28, no. 8, pp. 1–20, Aug. 2025, doi: 10.1007/s10586-025-05173-8.

[19] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Ethereum smart contract analysis tools: A systematic review," *IEEE Access*, vol. 10, no. 1, pp. 57037–57062, 2022, doi: 10.1109/ACCESS.2022.3169902.

[20] Y. Yao, H. Li, X. Yang, and Y. Le, "An improved vulnerability detection system of smart contracts based on symbolic execution," *IEEE International Conference on Big Data*, vol. 2022, no. Dec., pp. 3225–3234, Dec. 2022, doi: 10.1109/BigData55660.2022.10020730.

[21] Y. Xu et al., "A review of code vulnerability detection techniques based on static analysis," *Mechanisms and Machine Science*, vol. 2024, no. Jan., pp. 251–272, Jan. 2024, doi: 10.1007/978-3-031-44947-5_21.

[22] Z. Zheng, J. Su, J. Chen, D. Lo, Z. Zhong, and M. Ye, "DAppSCAN: Building large-scale datasets for smart contract weaknesses in DApp projects," *IEEE Transactions on Software Engineering*, vol. 50, no. 6, pp. 1360–1373, Jun. 2024, doi: 10.1109/TSE.2024.3383422.

[23] F. Jiang et al., "Enhancing smart-contract security through machine learning: A survey of approaches and techniques," *Electronics*, vol. 12, no. 9, pp. 1–20, Apr. 2023, doi: 10.3390/electronics12092046.

[24] J. Alghamdi, Y. Lin, and S. Luo, "A comparative study of machine learning and deep learning techniques for fake news detection," *Information*, vol. 13, no. 12, pp. 1–15, Dec. 2022, doi: 10.3390/info13120576.

[25] P. Guleria, "Interpretable learning models: An XAI-focused evaluation of classifier performance," *Neural Computing and Applications*, vol. 38, no. 6, pp. 1–20, Mar. 2026, doi: 10.1007/s00521-026-11933-3.