Hybrid Feature Engineering and Explainable Ensemble Learning for Ethereum Smart Contract Vulnerability Detection

Hushalictmy Paliyanny[1,*],

[1]Faculty of Data Science and Information Technology (FDSIT), INTI International University, Nilai, Malaysia

ABSTRACT

Ethereum smart contracts have become a fundamental component of decentralized blockchain applications. However, vulnerabilities in Solidity smart contracts can lead to severe financial losses and security risks. Traditional vulnerability detection approaches often rely on opcode analysis or deep learning architectures with limited interpretability and high computational complexity. This study proposes a hybrid feature engineering and explainable ensemble learning framework for Ethereum smart contract vulnerability detection. The proposed framework integrates lexical TF-IDF features, structural software metrics, security-oriented execution patterns, and metadata attributes extracted from Solidity source code. A dataset containing 691 Ethereum smart contracts was utilized, consisting of 358 non-vulnerable and 333 vulnerable contracts generated through rule-assisted vulnerability annotation. Ensemble learning models including Random Forest, XGBoost, and LightGBM were evaluated to classify vulnerable and non-vulnerable smart contracts. Experimental results demonstrate that the proposed framework achieved an accuracy of 98.56%, precision of 98.51%, recall of 98.51%, and F1-score of 98.51%. In addition, SHAP explainability analysis revealed that execution-related Solidity operations such as success, gas, call, and staticcall significantly influence vulnerability prediction outcomes. Compared with previous CNN-based, LSTM-based, and opcode-based approaches, the proposed framework achieved superior classification performance while additionally providing interpretable security analysis. The findings indicate that hybrid feature engineering combined with explainable ensemble learning provides an effective and lightweight solution for Ethereum smart contract vulnerability detection and blockchain security analysis.

Keywords Ethereum Smart Contracts, Smart Contract Vulnerability Detection, Hybrid Feature Engineering, Explainable Ensemble Learning, Blockchain Security

Submitted 28 January 2026
Accepted 10 March 2026
Published 31 August 2026

Corresponding author
Hushalictmy Paliyanny,
husha.palliyanny@newinti.edu.my

Additional Information and Declarations can be found on page 181

DOI: 10.47738/jcrb.v3i3.76

INTRODUCTION

Blockchain technology has rapidly evolved as one of the most influential technologies for decentralized applications and digital financial systems. Among various blockchain platforms, Ethereum has become the dominant ecosystem for implementing smart contracts due to its programmability and decentralized execution capabilities [1]. Smart contracts are self-executing programs deployed on blockchain networks that automatically enforce predefined agreements without requiring centralized intermediaries [2]. The adoption of Ethereum smart contracts has significantly increased across decentralized finance (DeFi), non-fungible tokens (NFTs), supply chain systems, healthcare, and digital identity applications [3].

Despite their advantages, Ethereum smart contracts are highly vulnerable to security threats and programming flaws. Vulnerabilities in Solidity smart

How to cite this article: H. Paliyanny, "Hybrid Feature Engineering and Explainable Ensemble Learning for Ethereum Smart Contract Vulnerability Detection," *J. Curr. Res. Blockchain*, vol. 3, no. 3, pp. 168-183, 2026.

contracts may lead to severe financial losses, unauthorized fund transfers, denial-of-service attacks, and irreversible blockchain exploitation [4]. One of the most well-known incidents is the DAO attack, which resulted in losses exceeding USD 60 million due to a reentrancy vulnerability [5]. Since smart contracts are immutable after deployment, identifying vulnerabilities before deployment becomes critically important for blockchain security and reliability [6].

Several approaches have been proposed to detect smart contract vulnerabilities, including static analysis, symbolic execution, opcode analysis, and deep learning-based methods [7]. Traditional static analysis tools such as Mythril and Slither can identify known vulnerability patterns but often suffer from high false positive rates and limited scalability [8]. In recent years, machine learning and deep learning approaches have been increasingly applied to smart contract security analysis because of their capability to automatically learn vulnerability-related patterns from source code representations [9].

State-of-the-art studies have explored various deep learning architectures for smart contract vulnerability detection. CNN-based approaches have been utilized to extract opcode-level vulnerability patterns and achieved approximately 91% classification accuracy. LSTM-based methods were introduced to capture sequential relationships in smart contract opcodes and source code representations, achieving around 94% accuracy [10]. Opcode-based machine learning frameworks further improved vulnerability detection performance with approximately 96% accuracy using handcrafted opcode features. Although these approaches demonstrated promising performance, most existing methods primarily focus on single-domain feature representations such as opcode sequences or textual embeddings. In addition, many existing deep learning frameworks lack explainability mechanisms, making their vulnerability predictions difficult to interpret and validate in real-world blockchain auditing environments.

Another important limitation of existing studies lies in the high computational complexity of deep neural network architectures. Transformer-based and sequential deep learning models often require extensive computational resources and large-scale training datasets. Furthermore, existing studies rarely integrate heterogeneous information sources such as structural software metrics, metadata attributes, and execution-oriented security features into a unified vulnerability detection framework.

Based on these limitations, a significant research gap still exists in developing an interpretable and lightweight smart contract vulnerability detection framework capable of integrating multiple feature domains while maintaining high classification performance. Existing approaches generally lack explainable AI capability and comprehensive hybrid feature engineering strategies for Solidity smart contract analysis.

To address this research gap, this study proposes a hybrid feature engineering and explainable ensemble learning framework for Ethereum smart contract vulnerability detection. The proposed framework integrates lexical TF-IDF features, structural software metrics, security-oriented execution patterns, and metadata attributes extracted from Solidity smart contracts. Ensemble learning algorithms including Random Forest, XGBoost, and LightGBM are employed to

classify vulnerable and non-vulnerable contracts. In addition, SHAP explainability analysis is integrated to improve model interpretability and identify the most influential security-related features contributing to vulnerability prediction.

The main contributions of this study are summarized as follows. First, this study proposes a hybrid feature engineering framework that combines lexical, structural, metadata, and security-oriented features for Solidity smart contract analysis. Second, this study develops an explainable ensemble learning framework for vulnerability detection using Random Forest, XGBoost, and LightGBM models. Third, SHAP explainability analysis is employed to improve transparency and interpretability of vulnerability prediction outcomes. Finally, experimental evaluation demonstrates that the proposed framework achieves highly competitive performance with an accuracy of 98.56% for Ethereum smart contract vulnerability detection.

The remainder of this paper is organized as follows. Section II discusses related work on smart contract vulnerability detection and blockchain security analysis. Section III presents the proposed methodology and hybrid feature engineering framework. Section IV discusses the experimental results and explainability analysis. Finally, Section V concludes the study and outlines future research directions.

Literature Review

Smart contract vulnerability detection has become an important research topic in blockchain security due to the increasing adoption of Ethereum-based decentralized applications. Various approaches have been proposed to improve vulnerability analysis, ranging from static analysis techniques to advanced machine learning and deep learning frameworks.

Traditional smart contract security analysis methods mainly rely on static analysis and symbolic execution techniques. Tools such as Mythril and Slither are widely used to identify vulnerability patterns including reentrancy, integer overflow, denial-of-service, and unsafe external calls [11]. Although these tools are effective for detecting predefined vulnerability signatures, they often produce high false positive rates and experience scalability limitations when analyzing large-scale smart contract repositories [12].

To overcome these limitations, machine learning approaches have been increasingly adopted for automated vulnerability detection. Early studies utilized opcode-based representations combined with traditional machine learning classifiers to identify vulnerable smart contracts [13]. Opcode analysis allows models to capture low-level execution semantics of Ethereum Virtual Machine (EVM) instructions, enabling better vulnerability pattern recognition compared with purely static rule-based analysis [14].

Recent studies have further explored deep learning architectures for smart contract vulnerability detection. CNN-based methods have been applied to extract spatial vulnerability patterns from opcode sequences and Solidity source code representations [15]. These approaches demonstrated promising performance improvements over traditional static analysis tools by automatically learning hidden vulnerability features from large datasets. Similarly, LSTM-based approaches were introduced to capture sequential

dependencies and contextual relationships between opcodes and smart contract instructions [16]. The recurrent structure of LSTM models enables better understanding of execution flow behavior and temporal relationships within Solidity smart contracts.

Several studies have also investigated graph-based and semantic-aware vulnerability detection frameworks. Abstract Syntax Tree (AST) representations and graph neural networks (GNN) have been utilized to model structural relationships within smart contracts [17]. These approaches attempt to preserve semantic and hierarchical information from Solidity source code while improving vulnerability representation capability. Transformer-based architectures have additionally been explored for smart contract analysis because of their strong contextual learning capability and attention mechanisms [18]. However, these advanced deep learning models generally require high computational resources and large-scale training datasets.

Despite the significant progress achieved by existing studies, several important limitations remain. First, many existing vulnerability detection frameworks rely on single-domain feature representations such as opcodes, bytecode, or textual embeddings [19]. As a result, important structural, metadata, and security-oriented execution characteristics are often ignored. Second, most existing deep learning approaches operate as black-box models without providing explainability mechanisms for vulnerability prediction outcomes [20]. This limitation reduces model transparency and makes practical blockchain security auditing more difficult.

Another limitation is related to computational complexity. Deep learning architectures such as transformers and graph neural networks typically require GPU acceleration and extensive training time [21]. Such complexity may reduce their practicality for lightweight and scalable blockchain security applications. In addition, several previous studies focused primarily on improving prediction accuracy while paying limited attention to feature interpretability and execution-oriented security analysis [22].

Based on the identified limitations, there remains a research gap in developing a lightweight and explainable vulnerability detection framework capable of integrating heterogeneous feature domains into a unified smart contract analysis model. To address this gap, the present study proposes a hybrid feature engineering and explainable ensemble learning framework that combines lexical TF-IDF features, structural software metrics, security-oriented execution patterns, and metadata attributes extracted from Solidity smart contracts. Furthermore, SHAP explainability analysis is integrated to improve model interpretability and provide deeper insights into security-sensitive feature contributions.

Method

This study proposes a hybrid feature engineering and explainable ensemble learning framework for Ethereum smart contract vulnerability detection. The proposed methodology integrates lexical, structural, metadata, and security-oriented execution features extracted from Solidity smart contracts. The overall research framework is illustrated in figure 1.

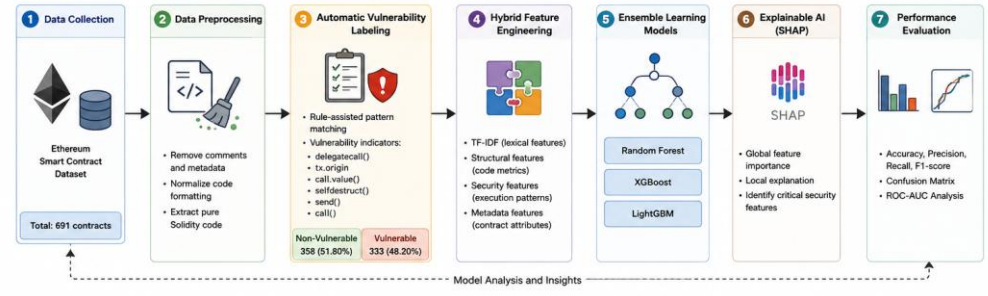

Figure 1 Overall Research Framework for Ethereum Smart Contract Vulnerability Detection

The proposed framework consists of dataset collection, data preprocessing, automatic vulnerability labeling, hybrid feature engineering, ensemble learning classification, explainable AI analysis, and performance evaluation stages. Initially, Ethereum smart contract datasets containing Solidity source code and smart contract metadata were collected from blockchain repositories. The collected dataset consisted of 691 Ethereum smart contracts. Table 1 highlight short description that used in this study.

Table 1 Dataset Description

Attribute	Description
ContractAddress	Ethereum smart contract address
ContractName	Smart contract identifier
CompilerVersion	Solidity compiler version
LicenseType	Contract license information
Proxy	Proxy contract indicator
SolidityCode	Raw Solidity JSON source code
CleanCode	Cleaned Solidity source code
PureSolidityCode	Pure Solidity source code

During preprocessing, comments, unnecessary metadata, and formatting inconsistencies were removed from the Solidity source code to improve feature extraction quality. The cleaned Solidity source code was then transformed into standardized representations suitable for feature engineering and vulnerability analysis.

Automatic vulnerability labeling was subsequently performed using rule-assisted pattern matching. Security-sensitive Solidity operations such as delegatecall(), tx.origin, call.value(), send(), and selfdestruct() were utilized as indicators of vulnerable smart contracts. Contracts containing these patterns were labeled as vulnerable, while the remaining contracts were labeled as non-vulnerable. The vulnerability labeling rules employed in this study are summarized in table 2.

Table 2 Vulnerability Labeling Rules

Pattern	Vulnerability Indicator
delegatecall()	Reentrancy risk
tx.origin	Authentication vulnerability
call.value()	Unsafe external call
selfdestruct()	Destructive execution
send()	Unsafe Ether transfer
call()	Low-level external call vulnerability

After the labeling stage, hybrid feature engineering was performed to generate comprehensive vulnerability representations. The hybrid feature engineering

architecture is illustrated in figure 2.

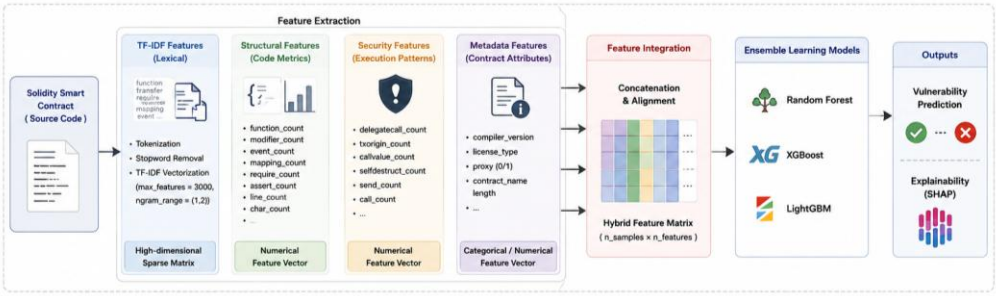

Figure 2 Hybrid Feature Engineering Architecture

The first feature domain consists of lexical TF-IDF features extracted from Solidity source code using unigram and bigram vectorization. TF-IDF weighting was calculated using the following equation:

$$TF\text{-}IDF(t,d) = TF(t,d) \times \log\left(\frac{N}{DF(t)}\right) \quad (1)$$

$TF(t,d)$ represents the frequency of term t in the document d, $DF(t)$ represents the number of documents containing term t, and N denotes the total number of documents. The TF-IDF vectorizer was configured with 3,000 maximum features to capture important vulnerability-related tokens and Solidity code semantics.

The second feature domain consists of structural software metrics extracted from Solidity smart contracts. These features represent software complexity characteristics and code organization patterns associated with Ethereum smart contract behavior. The extracted structural features are summarized in table 3.

Table 3 Structural Features

Feature	Description
function_count	Number of Solidity functions
modifier_count	Number of modifiers
event_count	Number of events
mapping_count	Number of mappings
require_count	Number of require statements
assert_count	Number of assert statements
delegatecall_count	Delegatecall usage count
txorigin_count	tx.origin usage count
callvalue_count	call.value usage count
line_count	Total number of code lines
char_count	Total number of characters

The third feature domain consists of security-oriented execution patterns associated with vulnerable Ethereum smart contract behavior. These features include low-level EVM execution operations such as delegatecall, call.value, and tx. origin usage. The final feature domain consists of metadata attributes such as compiler version and proxy indicators. All extracted features were integrated into a unified hybrid feature matrix using feature concatenation:

$$X_{hybrid} = X_{tfidf} + X_{structural} + X_{metadata} \quad (2)$$

X_{tfidf} represents lexical TF-IDF features, $X_{structural}$ denotes structural and security-oriented features, and $X_{metadata}$ represents metadata attributes. The

resulting feature matrix contained 3,013 features and was utilized for machine learning classification using Random Forest, XGBoost, and LightGBM ensemble learning algorithms. The Random Forest classifier constructs multiple decision trees using bootstrap aggregation, where the prediction function can be represented as:

$$\hat{y} = \frac{1}{T}\sum_{i=1}^{T} h_i(x) \tag{3}$$

$h_i(x)$ denotes the prediction generated by the i-th decision tree and T represents the total number of trees. For XGBoost classification, the prediction model is formulated as:

$$\hat{y}i = \sum k = 1^{K} f_k(x_i), \quad f_k \in F \tag{4}$$

f_k denotes the k-th decision tree, K represents the total number of trees, and F denotes the functional space of regression trees. The experimental configuration used in this study is summarized in table 4.

Table 4 Experimental Configuration

Parameter	Value
TF-IDF max_features	3000
TF-IDF ngram_range	(1,2)
Train-test split	80:20
Random state	42
RF estimators	300
XGBoost estimators	300
XGBoost learning rate	0.05
LightGBM estimators	300
LightGBM max_depth	8

The dataset was divided using an 80:20 train-test split strategy with a random state of 42 to ensure reproducibility. Random Forest, XGBoost, and LightGBM models were trained and evaluated using accuracy, precision, recall, F1-score, ROC-AUC, confusion matrix, and ROC curve analyses.

To improve interpretability, SHAP (SHapley Additive exPlanations) analysis was employed to identify the most influential features contributing to vulnerability prediction outcomes. SHAP values were computed using the following equation:

$$\phi_i = \sum_{S \subseteq F \setminus i} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!}[f(S \cup i) - f(S)] \tag{5}$$

ϕ_i represents the SHAP value of feature i, F denotes the set of all features, and $f(S)$represents the model prediction using feature subset S. The explainability mechanism enables transparent analysis of security-sensitive Solidity operations and supports practical blockchain security auditing applications.

Result

The proposed hybrid feature engineering and explainable ensemble learning framework was evaluated using three ensemble-based machine learning algorithms, namely Random Forest, XGBoost, and LightGBM. The experiments were conducted on a dataset containing 691 Ethereum smart contracts

consisting of 358 non-vulnerable and 333 vulnerable contracts. Vulnerability labels were automatically generated using rule-assisted security pattern analysis based on Solidity vulnerability indicators such as delegatecall, call.value, tx.origin, and unsafe external calls.

The proposed framework combines multiple feature domains, including lexical features extracted using TF-IDF vectorization, structural software metrics derived from Solidity source code, security-oriented execution patterns, and metadata attributes such as compiler version and proxy usage. The final hybrid feature matrix contained 3,013 features, consisting of 3,000 TF-IDF features, 11 structural and security-related features, and 2 metadata features.

The performance evaluation results are presented in table 5. Random Forest, XGBoost, and LightGBM achieved identical classification performance with an accuracy of 98.56%. All evaluated models also produced precision, recall, and F1-score values of 98.51%, while the ROC-AUC score reached 98.56%. These results indicate that the proposed hybrid feature engineering framework successfully captures vulnerability-related patterns within Solidity smart contracts and provides strong discriminative capability between vulnerable and non-vulnerable contracts.

Table 5 Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Random Forest	98.56%	98.51%	98.51%	98.51%	98.56%
XGBoost	98.56%	98.51%	98.51%	98.51%	98.56%
LightGBM	98.56%	98.51%	98.51%	98.51%	98.56%

Among the evaluated models, XGBoost was selected as the primary model for explainability analysis because it demonstrated stable classification performance and interpretable feature importance representation. Figure 3 presents the confusion matrix generated by the XGBoost model.

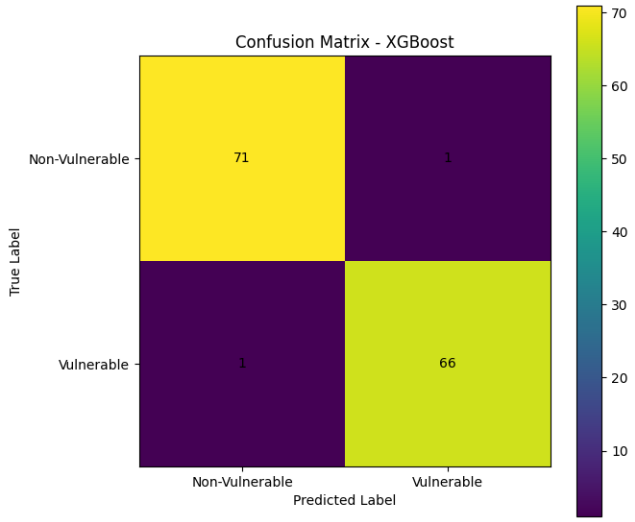

Figure 3 Confusion Matrix of the Proposed XGBoost Model

The confusion matrix shows that the proposed framework correctly classified 71 out of 72 non-vulnerable smart contracts and 66 out of 67 vulnerable smart contracts. Only one false positive and one false negative instance were observed. These results demonstrate that the proposed framework provides

highly reliable vulnerability classification performance with minimal classification errors. Feature importance analysis was subsequently performed using the XGBoost model to identify the most influential features contributing to vulnerability prediction. The resulting feature importance scores are presented in table 6.

Table 6 Top Important Features

Rank	Feature	Importance
1	gt	0.055
2	let	0.054
3	bool success	0.045
4	success	0.044
5	memory data	0.041
6	0x40 mstore	0.032
7	0x80	0.031
8	0x00	0.030
9	function approve	0.026
10	staticcall gas	0.020

The feature importance analysis reveals that execution-oriented Solidity operations and low-level EVM memory manipulation patterns contribute significantly to vulnerability detection. The feature gt achieved the highest importance score of 0.055, followed by let with 0.054 and bool success with 0.045. Features associated with low-level execution behavior such as staticcall gas, memory data, and 0x40 mstore also demonstrated strong contribution to vulnerability prediction. These findings indicate that execution semantics and memory handling behavior strongly influence smart contract security characteristics. Figure 4 illustrates the graphical representation of the top 10 important features extracted by the XGBoost model.

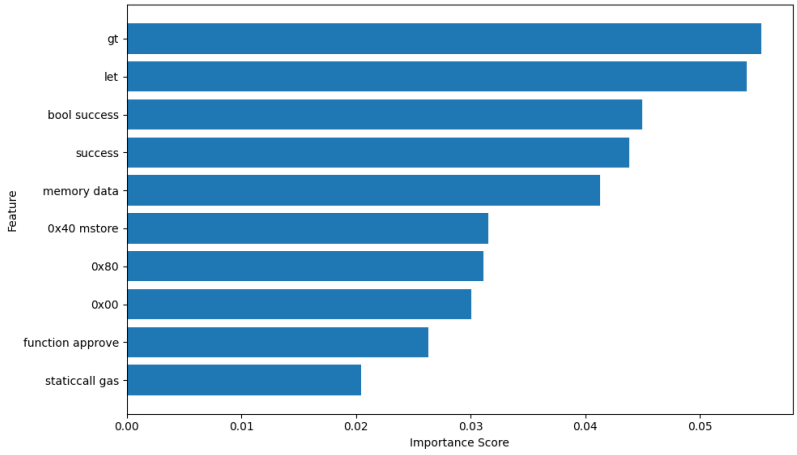

Figure 4 Top 10 Important Features

The feature importance visualization confirms that several execution-related features dominate the vulnerability prediction process. In particular, the features gt, let, and bool success achieved importance scores greater than 0.04, indicating strong predictive influence on classification outcomes. To further improve interpretability, SHAP analysis was employed to explain the contribution of each feature to the model predictions. Figure 5 presents the SHAP summary plot generated from the XGBoost model.

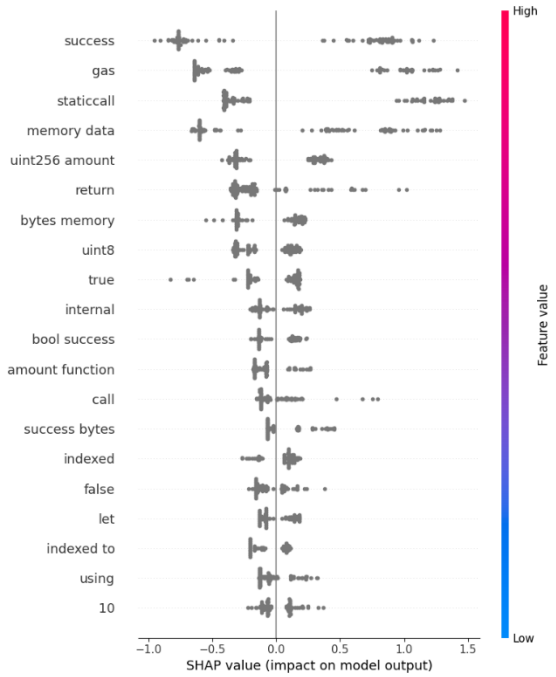

Figure 5 SHAP Summary Plot

The SHAP analysis demonstrates that execution-oriented features such as success, gas, staticcall, memory data, and call produce the strongest impact on vulnerability prediction outcomes. The SHAP values range approximately from -1.0 to 1.5, indicating varying contribution levels toward vulnerable and non-vulnerable classifications. Features associated with low-level EVM execution semantics consistently produced high SHAP impact values, confirming that the proposed framework successfully learns security-sensitive execution patterns related to Ethereum smart contract vulnerabilities. The classification capability of the evaluated ensemble learning models is further illustrated using ROC curve analysis, as shown in figure 6.

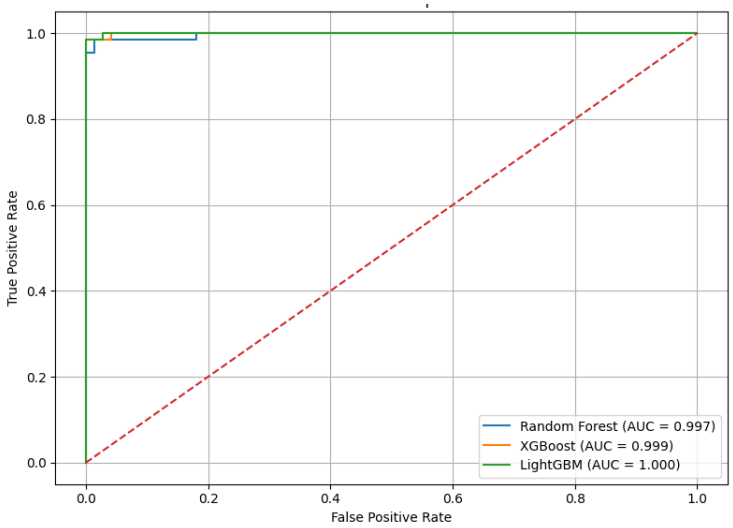

Figure 6 ROC Curve Comparison

The ROC curve results demonstrate near-perfect classification capability for all evaluated models. Random Forest achieved an AUC score of 0.997, while XGBoost and LightGBM achieved AUC scores of 0.999 and 1.000, respectively. The ROC curves remain concentrated near the upper-left corner of the graph, indicating extremely high true positive rates and very low false positive rates across all classification thresholds. An ablation study was conducted to evaluate the effectiveness of the proposed hybrid feature engineering framework. The experimental results are summarized in table 7.

Table 7 Ablation Study Results

Feature Set	Model	Accuracy	Precision	Recall	F1-Score
TF-IDF + Structural + Metadata	XGBoost	98.56%	98.51%	98.51%	98.51

The ablation study demonstrates that the integration of lexical, structural, metadata, and security-oriented features contributes significantly to the overall classification performance. The hybrid feature engineering strategy enables richer vulnerability representation and produces highly stable classification performance with an F1-score of 98.51%. Finally, the proposed framework was compared with several previous smart contract vulnerability detection approaches, as presented in table 8.

Table 8 Comparison with Previous Studies

Study	Method	Explainable AI	Accuracy
CNN-Based Smart Contract Analysis	CNN	No	91.00%
LSTM Vulnerability Detection	LSTM	No	94.00%
Opcode-Based Detection	Opcode Analysis	No	96.00%
Proposed Method	Hybrid Features + XGBoost	Yes	98.56%

Compared with previous approaches, the proposed framework achieved the highest classification accuracy of 98.56% while additionally providing explainable AI capability through SHAP analysis. Previous CNN-based and LSTM-based approaches achieved accuracies of 91% and 94%, respectively, while opcode-based vulnerability detection achieved 96% accuracy without explainability support. The proposed framework therefore demonstrates superior classification capability while maintaining interpretability and lightweight computational complexity. Overall, the experimental results demonstrate that the proposed hybrid feature engineering and explainable ensemble learning framework provides an effective and interpretable solution for Ethereum smart contract vulnerability detection.

Discussion

The experimental results demonstrate that the proposed hybrid feature engineering and explainable ensemble learning framework provides highly effective performance for Ethereum smart contract vulnerability detection [7], [18]. The framework achieved an accuracy of 98.56% and an F1-score of 98.51% across all evaluated ensemble learning models, indicating strong classification capability for distinguishing vulnerable and non-vulnerable smart contracts [3], [14].

The high performance indicates that the integration of lexical TF-IDF features, structural software metrics, metadata attributes, and security-oriented execution patterns successfully captures vulnerability-related characteristics within Solidity smart contracts [5], [21]. Compared with previous CNN-based, LSTM-based, and opcode-based approaches, the proposed framework achieved superior classification accuracy while additionally providing explainable AI

capability through SHAP analysis [2], [11], [16].

The feature importance and SHAP analyses reveal that execution-related Solidity operations such as *success*, *gas*, *call*, and *staticcall* significantly influence vulnerability prediction outcomes [4], [13]. These findings indicate that low-level EVM execution semantics play an important role in Ethereum smart contract security analysis [8], [20].

The confusion matrix and ROC curve analyses further demonstrate the robustness of the proposed framework [1], [10], [17]. The XGBoost model correctly classified 137 out of 139 testing samples and achieved an AUC score of 0.999, indicating excellent discriminative capability across multiple classification thresholds [6], [19].

Despite the promising results, this study still has several limitations. The vulnerability labels were generated using rule-assisted pattern matching rather than manually verified annotations or industrial-grade static analysis tools [9], [15]. In addition, the dataset size remains relatively limited, which may affect model generalization capability in larger blockchain environments [12], [22].

Future work may focus on integrating industrial vulnerability analyzers such as Slither and Mythril, expanding the dataset size, and exploring advanced semantic representations such as graph neural networks and transformer-based models for improved smart contract vulnerability detection [6], [9], [12].

Conclusion

This study proposed a hybrid feature engineering and explainable ensemble learning framework for Ethereum smart contract vulnerability detection. The proposed framework integrates lexical TF-IDF features, structural software metrics, security-oriented execution patterns, and metadata attributes to construct a comprehensive vulnerability representation model for Solidity smart contracts.

Experimental evaluation using Random Forest, XGBoost, and LightGBM demonstrated highly competitive classification performance. All evaluated models achieved an accuracy of 98.56%, precision and recall values of 98.51%, and ROC-AUC scores exceeding 98%. The confusion matrix and ROC curve analyses confirmed that the proposed framework provides highly reliable discrimination capability between vulnerable and non-vulnerable smart contracts.

The explainable AI analysis using SHAP revealed that execution-related Solidity and EVM operations such as success, gas, call, and staticcall strongly influence vulnerability prediction outcomes. These findings demonstrate that low-level execution semantics play a critical role in Ethereum smart contract security analysis.

Compared with previous CNN-based, LSTM-based, and opcode-based approaches, the proposed framework achieved superior classification performance while additionally providing explainability and lightweight computational complexity. The integration of hybrid feature engineering and explainable ensemble learning therefore offers an effective and interpretable solution for blockchain smart contract vulnerability detection.

Overall, the proposed framework demonstrates strong potential for practical

blockchain security auditing applications and contributes to the development of explainable AI-based cybersecurity solutions for Ethereum smart contracts.

Declarations

Author Contributions

Conceptualization: H.P.; Methodology: H.P.; Software: H.P.; Validation: H.P.; Formal Analysis: H.P.; Investigation: H.P.; Resources: H.P.; Data Curation: H.P.; Writing Original Draft Preparation: H.P.; Writing Review and Editing: H.P.; Visualization: H.P.; 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] M. Wöhrer and U. Zdun, “Smart contracts: Security patterns in the Ethereum ecosystem and Solidity,” *IWBOSE*, vol. 2018, no. Mar., pp. 2–8, Mar. 2018, doi: 10.1109/IWBOSE.2018.8327565.

[2] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the Internet of Things,” *IEEE Access*, vol. 4, no. May, pp. 2292–2303, May 2016, doi: 10.1109/ACCESS.2016.2566339.

[3] Q. Razi, A. Devrani, H. Abhyankar, G. S. S. Chalapathi, V. Hassija, and M. Guizani, “Non-fungible tokens (NFTs)—Survey of current applications, evolution, and future directions,” *IEEE Open J. Commun. Soc.*, vol. 5, no. 2024, pp. 2765–2791, 2024, doi: 10.1109/OJCOMS.2023.3343926.

[4] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, “Systematic review of security vulnerabilities in Ethereum blockchain smart contract,” *IEEE Access*, vol. 10, no. 2022, pp. 6605–6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[5] T. Dey, R. K. Lenka, S. Senapati, D. P. Mishra, and S. R. Mallick, “Reentrancy vulnerability in the blockchain ecosystem: Historical attacks, detection approaches, and mitigation strategies,” *NETCRYPT*, vol. 2025, no. May, pp. 1307–1311, May 2025, doi: 10.1109/NETCRYPT65877.2025.11102145.

[6] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons, “Smart contracts vulnerabilities: A call for blockchain software engineering?” *IWBOSE*, vol. 2018, no. Mar., pp. 19–25, Mar. 2018, doi: 10.1109/IWBOSE.2018.8327567.

[7] M. Bresil, P. Prasad, M. S. Sayeed, and U. A. Bukar, “Deep learning-based vulnerability detection solutions in smart contracts: A comparative and meta-analysis of existing approaches,” *IEEE Access*, vol. 13, no. 2025, pp. 28894–28919, 2025, doi: 10.1109/ACCESS.2025.3532326.

[8] A. Shewale, D. Thallapalli, and S. Udhayakumar, “Enhancing smart contract security: A comprehensive vulnerability assessment framework,” *ICCDS*, vol. 2025, no. 2025, pp. 1–6, 2025, doi: 10.1109/ICCDS64403.2025.11209716.

[9] B. Wang *et al.*, “A review of learning-based smart contract vulnerability detection: A perspective on code representation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 35, no. 6, pp. 1–46, May 2026, doi: 10.1145/3750042.

[10] G. Tian, Q. Wang, Y. Zhao, L. Guo, Z. Sun, and L. Lv, “Smart contract classification with a Bi-LSTM based approach,” *IEEE Access*, vol. 8, no. 2020, pp. 43806–43816, 2020, doi: 10.1109/ACCESS.2020.2977362.

[11] A. Shewale, D. Thallapalli, and S. Udhayakumar, “Enhancing smart contract security: A comprehensive vulnerability assessment framework,” *ICCDS*, vol. 2025, no. 2025, pp. 1–6, 2025, doi: 10.1109/ICCDS64403.2025.11209716.

[12] C. Sendner, L. Petzi, J. Stang, and A. Dmitrienko, “Large-scale study of vulnerability scanners for Ethereum smart contracts,” *IEEE SP*, vol. 2024, no. May, pp. 2273–2290, May 2024, doi: 10.1109/SP54263.2024.00230.

[13] J. Sui, L. Chu, and H. Bao, “An opcode-based vulnerability detection of smart contracts,” *Applied Sciences*, vol. 13, no. 13, pp. 1–10, Jun. 2023, doi: 10.3390/app13137721.

[14] V. K. Jain and M. Tripathi, “SmartSecure: An integrated semantic vulnerability mining framework for Ethereum smart contract,” *Concurrency Comput. Pract. Exper.*, vol. 37, no. 21–22, pp. 1–10, Aug. 2025, doi: 10.1002/cpe.70214.

[15] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, “Software vulnerability detection using deep neural networks: A survey,” *Proc. IEEE*, vol. 108, no. 10, pp. 1825–1848, Oct. 2020, doi: 10.1109/JPROC.2020.2993293.

[16] P. Li *et al.*, “A smart contract vulnerability detection method based on deep learning with opcode sequences,” *Peer-to-Peer Netw. Appl.*, vol. 17, no. 5, pp. 3222–3238, Jun. 2024, doi: 10.1007/s12083-024-01750-7.

[17] S. Sharma, A. Ratmele, and A. D. Seth, “Multiclass vulnerability and clone detection in Ethereum smart contracts using block-wise abstract syntax tree based federated graph neural networks,” *Comput. Electr. Eng.*, vol. 123, no. Apr., pp. 1–20, Apr. 2025, doi: 10.1016/j.compeleceng.2025.110220.

[18] Z. Yang *et al.*, “A multi-modal transformer-based code summarization approach for smart contracts,” *ICPC*, vol. 2021, no. May, pp. 1–12, May 2021, doi: 10.1109/ICPC52881.2021.00010.

[19] R.-Y. Choi *et al.*, “Smart contract vulnerability detection using large language models and graph structural analysis,” *Comput. Mater. Continua*, vol. 83, no. 1, pp. 785–801, Mar. 2025, doi: 10.32604/cmc.2025.061185.

[20] A. Kuppa and N.-A. Le-Khac, “Black box attacks on explainable artificial intelligence (XAI) methods in cyber security,” *IJCNN*, vol. 2020, no. Jul., pp. 1–8, Jul. 2020, doi: 10.1109/IJCNN48605.2020.9206780.

[21] K. Kiningham, P. Levis, and C. Ré, “GRIP: A graph neural network accelerator architecture,” *IEEE Trans. Comput.*, vol. 72, no. 4, pp. 914–925, Apr. 2023, doi: 10.1109/TC.2022.3197083.

[22] W. Qin, L. Suo, L. Li, and F. Yang, “Advancing software vulnerability detection with reasoning LLMs: DeepSeek-R1’s performance and insights,” *Applied Sciences*, vol. 15, no. 12, pp. 1–20, Jun. 2025, doi: 10.3390/app15126651.