IEEE ICBC'24 1570982019

Blockchain Smart Contract Vulnerability Detection and Segmentation Using ML and XAI

***Abstract*—In the realm of blockchain technology, it is common for vulnerabilities to be present in poorly designed blockchain smart contracts, making them an easy target for network attacks. These vulnerabilities exploit flaws in the code to manipulate the smart contract to their benefit, which could lead to significant financial losses. It is important to identify such vulnerabilities to reduce the risk of network attacks. Machine learning algorithms (ML) can detect the various types of vulnerabilities in smart contracts, but they lack explainability and are incapable of identifying the features responsible for a particular vulnerability type. Explainable artificial intelligence (XAI) methods play a large role in interpreting and identifying the features responsible for ML outputs. In this paper, SHAP and LIME, two well-known XAI approaches, have been utilized to discover the most critical features contributing to a particular vulnerability type of smart contracts. Then, smart contract code segments linked to these features have been identified. By linking these features to specific parts of the code, we can reduce vulnerability and strengthen smart contracts' security. We have used multiple metrics to measure the XAI models' performance, such as relevance, accuracy, and consistency. These metrics help us choose the best XAI model to use.**

***Index Terms*—Blockchain, Smart Contract, Vulnerability, Explainable AI, Machine Learning**

I. Introduction

Blockchain is a technology that enables decentralized procedures to support transactions. It utilizes a distributed ledger system ensuring immutability throughout the process. It has gained traction across industries, including finance, healthcare, energy grid management, supply chain operations, agriculture, and more [5]. Smart contracts are a part of this technology and function as self-executing programs that enforce contractual terms between multiple parties without the need for intermediaries like banks or lawyers [10]. By eliminating intermediaries, smart contracts offer time and cost savings while minimizing the risk of fraud or mistakes. However, it's crucial to acknowledge that smart contracts are susceptible to network attacks caused by coding errors. These vulnerabilities have led to losses in different sectors. For instance, the DAO's smart contract vulnerability resulted in a loss of $60 million in 2016 [17]. That's why it is essential to examine and ensure the security of contracts prior, to their inclusion, on the blockchain.

There are types of vulnerabilities in contracts, such as reentrancy attacks, timestamp dependencies, and infinite loop vulnerabilities. These vulnerabilities occur due to coding errors, design flaws, and logical mistakes. Smart contracts are typically written by developers with varying levels of experience, which can lead to errors and omissions [18]. Unlike software programs that can be updated to fix vulnerabilities in versions, smart contracts on blockchain technology are immutable once implemented on the network. However, this immutability exposes them to attacks because they store and manage cryptocurrencies on blockchain platforms.

Using machine learning (ML) and Deep learning (DL) models to detect vulnerabilities in contracts face challenges that need to be addressed [12]. One major challenge is the availability of labeled data required for training ML models to detect vulnerabilities [1], [15]. Although ML-based or DL-based approaches can accurately detect vulnerabilities in contracts, they lack the ability to explain or reason due to their black-box nature. In this scenario, explainable artificial intelligence (XAI) emerges as an option since it can provide explanations for the results obtained from ML or DL classifiers [3], [6], [24]. XAI is capable of determining the significance of features behind these results. Ultimately, XAI models can help answer questions such as: Why are certain outputs produced? Why are others not produced? Is there any bias present? Incorporating XAI models, like LIME [19] or SHAP [16], into Machine Learning or Deep Learning can offer more understandable explanations for detecting vulnerabilities in contracts. Therefore, this research paper showcases a system that effectively detects vulnerabilities in contracts by combining graph-based and ML approaches, providing insights. Moreover, the integrated XAI models with ML help identify the features responsible for vulnerabilities in smart contracts. Moreover, the system has the ability to identify the sections of code that are accountable, for particular kinds of vulnerabilities. In order to give you an insight into the paper's contents, Section 2 presents a review of existing literature, whereas Section 3 elaborates on the methodology utilized in this study. The results are outlined in Section 4, and, finally, Section 5 concludes the paper.

II. Literature Review

The topic of security risks in smart contracts has drawn considerable attention due to large financial losses caused by vulnerabilities. Numerous strategies for identifying these vulnerabilities have been proposed in recent years.

Jiang et al. [9] developed ContractFuzzer, a testing tool specifically designed for smart contracts on the Ethereum blockchain to pinpoint security weaknesses. The fuzzer generates fuzzing inputs based on the Application Binary Interface (ABI) specifications of smart contracts. It defines test oracles to detect security vulnerabilities. Likewise, Huang et al. [7] constructed a vulnerability detection model rooted in multi-task learning. The model uses auxiliary tasks to learn more

directional vulnerability features. AFS (AST Fuse program Slicing) is another tool introduced by Wang et al. [21] to combine unique code traits to learn about new vulnerabilities.

Additionally, some standard forms of vulnerabilities are identifiable through methods like symbolic execution and constraint solving, and pre-training techniques applied to control flow graphs [13] and vital data flow graphs [22]. However, these methods are limited in their capacity to extend their findings or locate the specific sections of flow graphs that correspond to vulnerabilities.

On the other hand, techniques such as Bi-LSTM, Attention, Deep-Cross Networks, and Convolutional Neural Networks (CNN) have been used in smart contract vulnerability detection to improve detection accuracy and scalability. Bi-LSTM and attention [20] mechanisms are used to capture the semantic and contextual information of the source code, which traditional static or dynamic analysis tools often miss. Deep-Cross Network and CNN can learn high-level features from the source code, making them effective in detecting complex vulnerabilities. For example, Li et al [11] proposed a modular vulnerability detection model called Link-DC to detect smart contract vulnerabilities. Link-DC captures richer feature information through stitching. This method employed contract graphs and pattern features as input and constructed low-dimensional and sparse features into high-dimensional nonlinear features using a deep and cross-network. Similarly, Wu et al. [23] proposed a hybrid attention mechanism (HAM) model to detect security vulnerabilities in smart contracts. HAM extracted code snippets from the source code that concentrated on critical vulnerability areas. Zhang et al [25] proposed a method that used a Bi-LSTM neural network to vectorize smart contract code. Vectorization allowed the detection of smart contract vulnerabilities. Hwang et al [8] exploited the resemblance between contract bytecode and RGB byte values in images to transform smart contracts into one-dimensional or two-dimensional images. This transformation led to a loss in sequential execution and interconnected relationships in the JUMP statements within the smart contracts. The approach was effective for identifying vulnerabilities using CNN on these converted images. These techniques and others suffer from labeled data scarcity and increased computational costs of deep learning models, which limits their practical applicability.

In summary, several approaches exist for identifying vulnerabilities in smart contracts, including formal verification, intermediate representation, symbolic execution, pattern matching, fuzzing, machine learning, and deep learning. Despite this, current detection systems either only recognize the existence of vulnerabilities without specifying their types, or rely on manually-engineered heuristics that aren't easily transferable across different vulnerability types. These methods also lack consideration for identifying features or code segments specifically responsible for vulnerabilities in smart contracts. Consequently, the next section introduces a methodology aimed at identifying both smart contract vulnerabilities and the corresponding code segments.

III. Methodology

Figure 1 shows the framework for detecting vulnerabilities and segmentation in smart contracts. The first section elaborates on the multiple stages involved in data processing. The next section goes over the machine learning techniques used for detecting vulnerabilities in smart contracts. Then, the following section covers the Explainable AI (XAI) methods employed for identifying features responsible for specific vulnerabilities. The last section elaborates on the segmentation processes for pinpointing code segments linked to particular vulnerabilities.

A. Data Processing

In the data processing pipeline for Ethereum smart contracts, several key steps are undertaken. Initially, Solidity code is compiled into EVM bytecode, and opcodes are then extracted for further analysis and vulnerability detection. This leads to the creation of a Control Flow Graph (CFG) to map the contract's execution flow. Random Biased Walk (RBW) is employed on the CFG to explore the code, after which the path is flattened into a sequence of opcodes suitable for machine learning. Tokenization breaks these opcodes into basic blocks, followed by feature selection, which isolates important attributes to optimize machine learning performance and resource use.

The second aspect involves fine-tuning the data for machine learning or deep learning models. To control the length of RBWs, feature selection algorithms are used, enhancing the performance by reducing overfitting and improving signal-to-noise ratio. Term Frequency-Inverse Document Frequency (TF IDF) [2] is employed as the final step to normalize the selected opcodes, ensuring that the most relevant features are highlighted. This comprehensive approach aims to improve the accuracy of smart contract vulnerability detection while maintaining computational efficiency.

B. Vulnerability Classification

To identify flaws, a classification algorithm is needed for sorting smart contracts into different types of vulnerabilities. This is done using machine learning (ML) and deep learning (DL) algorithms. The selection of the algorithm and its parameters hinges on the dataset's attributes and the desired performance metrics. In the following sections, we will delve into the specific ML and DL algorithms employed herein for classification in the detection of smart contract vulnerabilities.

1) Machine Learning Classification: Following the TFIDF pre-processing stage, various Machine Learning algorithms can be used for detecting vulnerabilities in smart contracts. Linear models like Logistic Regression, Linear Regression, Stochastic Gradient Descent, Ridge Classifier, and Passive Aggressive Classifier are suitable for binary classification tasks. Decision Tree and Random Forest classifiers are effective for both binary and multi-class classification problems. Ensemble techniques like AdaBoost, Extra Trees, and Bagging classifiers are popular for their capacity to lower variance and enhance

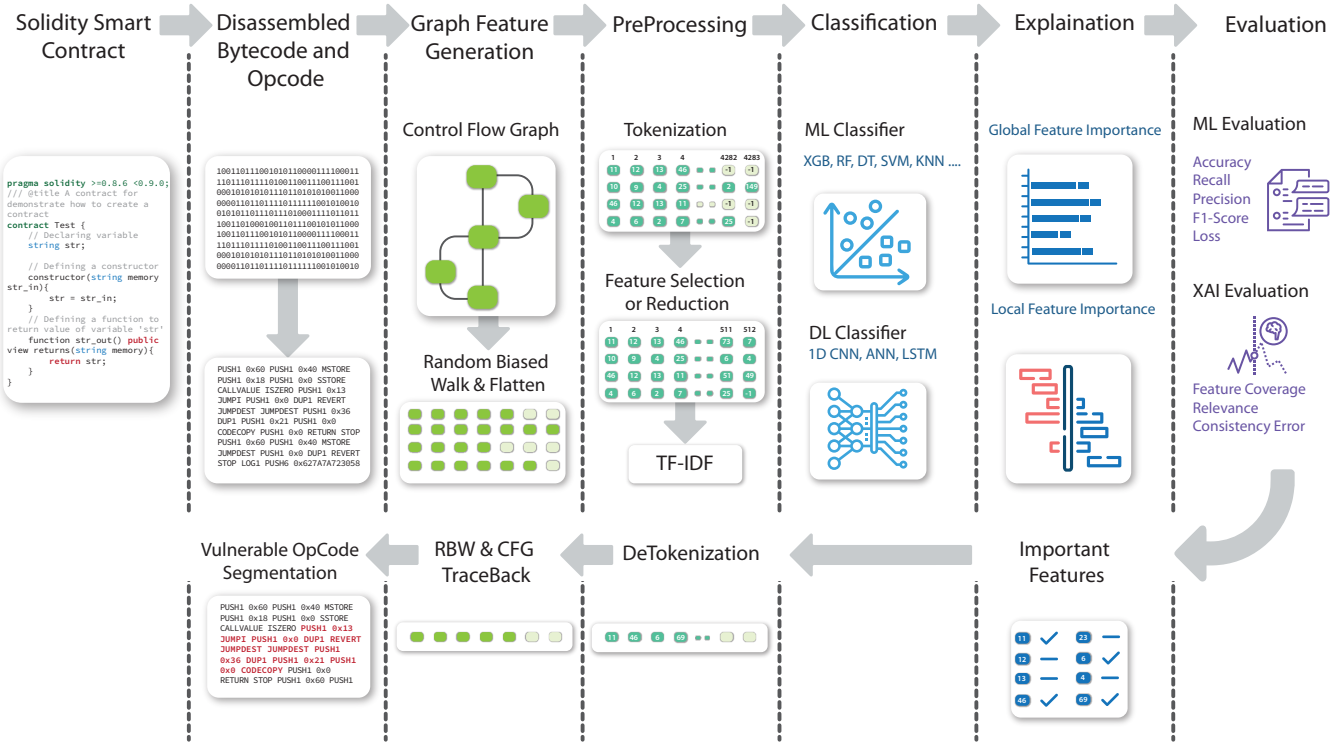

Fig. 1. Workflow for Smart Contract Vulnerability Detection using RBW and SHAP/LIME

model stability. Support Vector Classifier, Linear Support Vector Classifier, and k-Nearest Neighbor Classifier excel in high-dimensional datasets. Naive Bayes classifiers, including Gaussian, Bernoulli, Categorical, Complement, and Multinomial, are probabilistic models suitable for text classification tasks. Lastly, neural network-based classifiers such as the Multilayer Perceptron classifier can model intricate relationships and yield high accuracy. The selection of an algorithm is based on the problem's nature, dataset dimensions, and the targeted accuracy.

2) Deep Learning Classification: In addition to conventional ML algorithms, deep learning methods such as artificial neural networks (ANN), 1D convolutional neural networks (CNN), and CNN with long short-term memory (LSTM) are employed for classifying smart contract vulnerabilities. These algorithms have the ability to automatically learn high-level features from raw input data, eliminating the need for manual feature engineering. They are also adept at handling intricate relationships and patterns within the data, making them suitable for smart contract vulnerability detection. The ANN consists of four compact layers with 256, 1024, 128, and 64 neurons respectively. The input layer utilizes ReLU and is 512-dimensional, while the output layer contains 8 neurons and employs a sigmoid function. After each dense layer, batch normalization and dropout are applied to mitigate overfitting. The 1D CNN model is made up of three sets of convolutional layers, max-pooling layers, dropout, and batch normalization. The first set has 32 ReLU-activated size-3 filters, and the second and third sets have 64 filters each. Max-pooling layers reduce the dimensionality of the output feature maps, and the flattening layer converts the output to a 1D vector. Following this, dense layers with 128 and 64 neurons use ReLU and sigmoid activation functions respectively. Like the prior model, this one uses the categorical cross-entropy loss function, the Adam optimizer, and various metrics. Dropout and batch normalization are also included to prevent overfitting.

C. Explanation

SHAP: Merely identifying vulnerabilities in smart contracts isn't sufficient; pinpointing the features responsible for these vulnerabilities is also crucial. This is where Explainable Artificial Intelligence (XAI) models come into play. Among various XAI models, "SHAP" and "LIME" stand out for their agnostic nature, allowing them to be integrated with any kind of ML algorithm. "SHAP" is an acronym for "Shapley Additive exPlanations." It assigns a value to each input feature in an ML model to explain its contribution to the model's prediction or output. This value is known as the Shapley value, and the SHAP model is based on it. The Shapley value for a feature is essentially the average difference across all possible outcomes. Combining the Shapley values of all features provides a comprehensive explanation of the model's output that aligns with the original model. The Shapley values are calculated using the following equation [4]:

$$SHAP_i = \frac{1}{n} \sum_{S \subseteq F \setminus i} \frac{|S|!(n-|S|-1)!}{n!} (f(S \cup i) - f(S)) \quad (1)$$

where n represents the total number of input features, F is the set of input features, S is a subset of F, i denotes the feature of interest, and f is the prediction function. Any Machine Learning model can serve as the prediction function. $SHAP_i$ yields the needed Shapley value for a feature of interest.

Lime LIME serves as a technique for explaining predictions, irrespective of the model's kind or complexity. Standing for Local Interpretable Model-agnostic Explanations, LIME is designed to clarify a small portion of the model's behavior around a specific input and is versatile enough to be applied to any kind of model.

To predict the outcomes for new data points created near the important input, LIME employs any black box model, be it Machine Learning or Deep Learning. These data points are then weighted based on their closeness to the original input, with the most similar points getting the highest weight. LIME uses these weighted data points to construct a straightforward linear model, and its coefficients serve to explain the black box model's predictions. The underlying idea of LIME is that a basic linear model in a localized region around a specific input can approximate the behavior of a more complex model. This linear model helps to identify which features hold the most weight in the model's predictions and is easier to understand and interpret.

D. Segmentation

Explainability is vital in machine learning models, particularly in sensitive use-cases like identifying smart contract vulnerabilities. In this research, both SHAP and LIME frameworks were employed to explain the predictions made by ML and DL algorithms. The SHAP framework offers a roster of important features for a specific data point input and sorts these features based on their influence on the model's outcome. To evaluate the significance of these features, Shapley values from cooperative game theory are used.

The proposed opcode segmentation technique seeks to pinpoint the operation codes that are culpable for smart contract vulnerabilities. This is done by leveraging explainable AI along with a classifier. Initially, the Shapley value of the input solidity data is calculated to compile a list of noteworthy features and arrange them based on their impact on the classifier's output. Employing a pre-trained and stored tokenizer, the most crucial features are selected, and their associated tokens are fetched. This token is then detokenized to acquire the basic block, a sequence of opcodes.

To identify the opcode causing the vulnerability, RBW and CFG traceback processes are essential. This traceback confirms that the list of opcodes (basic block) obtained from detokenization aligns with the RBW within the CFG. The RBW serves to navigate through the CFG, representing the control flow of the smart contract. Tracing back from the basic block to the CFG allows for determining the path the basic block takes through the CFG. This traced path can then be used to identify the vulnerable opcode.

IV. Experimental Analysis

This section outlines the experimental setup employed to gauge the efficacy of our suggested approach for vulnerability detection in smart contracts. Metrics such as accuracy, precision, recall, F1-score, and the area under the receiver operating characteristic curve (AUC-ROC) are utilized for evaluation. Feature selection was performed using methods like Generic Univariate, Genetic Algorithm, Particle Swarm Optimization, L1-based, and Tree-based feature selection. Vulnerability classification in smart contracts is executed using an array of machine learning (ML) and deep learning (DL) algorithms. Additionally, explainable AI techniques like SHAP and LIME are applied to interpret the classification outcomes and evaluate the importance of the chosen features. Segmentation of opcodes was successfully conducted to pinpoint code sections that are susceptible to vulnerabilities. Subsequent sections will delve into the detailed analysis of the experimental results.

A. Dataset

TABLE I
SUMMARY OF VULNERABILITIES IN A SMART CONTRACT

Vulnerability Type	Number of Occurrences
Strict equality (SE)	1098
Block Number Dependency (BN)	3518
Dangerous delegatecall (DE)	291
Ether frozen (EF)	291
Integer overflow (OF)	1770
Reentrancy (RE)	1218
Timestamp dependency (TP)	936
Unchecked external call (UC)	3393

This research utilizes a dataset [14] that contains 12,515 smart contracts, each paired with its respective source code. The dataset features eight unique types of vulnerabilities, specifically timestamp dependency (TP), block number dependency (BN), dangerous delegate call (DG), Ether frozen (EF), unchecked external call (UC), reentrancy (RE), integer overflow (OF), and dangerous Ether strict equality (SE) as outlined in table I. These vulnerabilities are identified using a combination of vulnerability-specific patterns and manual inspections.

B. Assessment Metrics

1) ML Assessment Criteria: The performance of ML algorithms was assessed using multiple evaluation metrics, including accuracy, binary cross-entropy loss, recall, precision, F1-score, and AUC (area under the curve).

2) XAI Assessment Criteria: Metrics like feature importance, relevance, and consistency error are employed to gauge the quality and efficacy of the explanations produced by explainable AI. These evaluation metrics are essential for comparing, contrasting, and refining Explainable AI methods.

TABLE II
ML EVALUATION MEASURES

ML	Accuracy	F1-Score	TestAccuracy	Test F1-Score
Logistic Regression Classifier	0.5258	0.59	0.4880	0.54
Linear Regression Classifier	0.3043	0.02	0.3058	0.03
Stochastic Gradient Descent	0.5811	0.27	0.5407	0.22
Ridge Classifier	0.6153	0.31	0.5566	0.26
Passive Aggressive Classifier	0.6695	0.42	0.6198	0.33
Decision Tree Classifier	0.9990	1	0.9984	1
K-Nearest Neighbor Classifier	0.9983	1	0.9911	1
Radius Neighbor Classifier	0.3307	0.04	0.3198	0.03
Nearest Centroid Classifier	0.3835	0.3	0.3589	0.27
Random Forest Classifier	0.9904	0.97	0.9868	0.97
AdaBoost Classifier	0.3661	0.02	0.3578	0.01
Extreme Gradient Boosting Classifier	0.9990	1	0.9984	1
Hist Gradient Boosting Classifier	0.9990	1	0.9984	1
Bagging Classifier	0.9988	1	0.9988	1
Extra Trees Classifier	0.9990	1	0.9984	1
Support Vector Classifier	0.8931	0.69	0.8717	0.68
Linear Support Vector Classifier	0.6952	0.44	0.6333	0.36
Gaussian Naive Bayes Classifier	0.3951	0.25	0.3775	0.22
Bernoulli Naive Bayes Classifier	0.2746	0	0.2624	0
Categorical Naive Bayes Classifier	0.2746	0	0.2624	0
Complement Naive Bayes Classifier	0.3730	0.02	0.3694	0.02
Multinomial Naive Bayes Classifier	0.3941	0	0.3922	0
Multi-layer Perceptron classifier	0.9867	0.96	0.9841	0.96

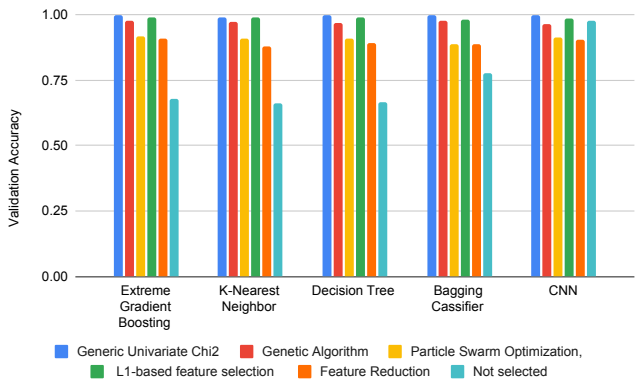

Fig. 2. Feature Selection Comparision using validation Accuracy

In evaluating explainable AI (XAI) models, three key metrics are commonly used. First is Feature Coverage [16], calculated by Equation $FC = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - \hat{y}_i^*|}{\hat{y}_i}$, which measures the proportion of dataset features explained by the XAI model. A higher value indicates better feature coverage. Second, Relevance is assessed through the equation $R = \sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{\bar{y}}$, which gauges the applicability of the XAI model's explanations. A higher relevance score is preferable. Lastly, Consistency Error evaluates the model's reliability using the formula $Error = \frac{1}{N+M} \times \sum_{i=1}^{N}(f(x_i) - g(x_i))$, where N and M are the sizes of two equally-sized dataset subsets. Lower consistency error values suggest the model is more reliable.

C. ML Results

Figure 2 shows that feature selection methods significantly impact the performance of classifiers. Among all classifiers, the Generic Univariate Chi2 feature selection technique achieves the highest validation accuracy. In this context, "not selected methods" pertain to using all features with padding. Feature reduction is simply cropping the first 512 features from the dataset.

Genetic Algorithm and Particle Swarm Optimization achieve lower validation accuracy compared to other feature selection methods like Generic and L1-Based. This could be attributed to whether irrelevant features are included or omitted during optimization. The L1-based feature selection method attains high accuracy levels that are equal to or exceed 0.98 across all classifiers. Feature Reduction registers the second-lowest validation accuracy among all classifiers, indicating it might not be effective in selecting the most pertinent features for the classification task. The Not chosen technique, which forgoes feature selection, attains relatively high accuracy levels with the CNN but shows subpar performance with other classifiers. This suggests that the CNN Classifier may either be less impacted by feature selection or that the selected features might not hold as much importance for this particular classifier.

Table II displays the performance of various machine learning algorithms for detecting vulnerabilities in smart contracts. The data reveals that some classifiers like decision trees, k-nearest neighbors, extreme gradient boosting, hist gradient boosting, bagging, and extra trees achieve flawless accuracy. Conversely, other methods, such as logistic regression, linear regression, stochastic gradient descent, ridge classifier, and support vector classifier, exhibit accuracy levels between 0.5 and 0.9. The decision tree classifier, k-nearest neighbor classifier, and extreme gradient boosting classifier register the highest F1 scores. The lowest F1 scores are recorded by linear regression, Bernoulli naive Bayes classifier, and multinomial naive Bayes classifier. Due to their ensemble

learning nature, Extreme Gradient Boosting, Decision Tree, K-Nearest Neighbor, Hist Gradient Boosting, Bagging, and Extra Tree outperform the rest, achieving both accuracy and F1-scores near 0.99.

D. DL results

The results illustrate the performance of an artificial neural network (ANN) in identifying vulnerabilities in smart contracts across various epochs. The results indicate a consistent decline in the loss function and a simultaneous increase in both accuracy and F1-score. This suggests an enhancement in the model's performance over time, applicable to both training and validation. The results indicate a loss reduction from 0.56 to 0.01, an accuracy boost from 0.22 to 0.98, and an F1-score rise from 0.24 to 0.99. Additionally, the data reveals a steady growth in the area under the receiver operating characteristic curve (AUC), from 0.61 to 0.99, which signals the model's capability to differentiate between positive and negative samples. The experiment confirms the efficacy of the proposed approach in detecting vulnerabilities in smart contracts using artificial neural networks.

The 1D convolutional neural network (1DCNN) results demonstrate that the 1DCNN employed in the experiment exhibited significant enhancements in performance over the course of multiple epochs. The first period had a modest accuracy of 0.28 and a considerable loss of 1.98. After 100 iterations, the loss was successfully decreased to 0.01, while the accuracy was significantly improved to 0.99. The metrics, namely recall, precision, and F1-score, exhibited steady improvement throughout the epochs, indicating the model's competence in accurately categorizing vulnerabilities in smart contracts. The ROC curve's area increased from 0.64 during the initial epoch to 0.99 in the concluding epoch, providing additional evidence of the enhanced performance of the model. The F1-score for validation increased significantly from 0.63 to 0.88, suggesting that the model demonstrates strong generalization capabilities when applied to previously unexplored data. In conclusion, the findings indicate that the utilization of the 1DCNN methodology proves to be a proficient approach to the detection and identification of vulnerabilities within smart contracts.

E. XAI Results

Figure 3 presents the SHAP values for each feature linked to a specific smart contract vulnerability, acquired via SHAP, enabling a comprehensive explanation of the ML model's outcomes. Conversely, Figure 4 illustrates the feature significance, generated using XGBClassifier, serving as a benchmark against the corresponding SHAP values. The "x" axis in Figure 3 marks the average SHAP value, while the "y" axis displays the feature list—generated during data processing through tokenization, elaborated in the methodology section. The "x" axis in Figure 4 represents feature importance as assessed by XGBClassifier, with the "y" axis showing the feature list. Figure 3 reveals that feature "X138" significantly influences the "Dangerous Delegate Call (DE)" vulnerability, with an

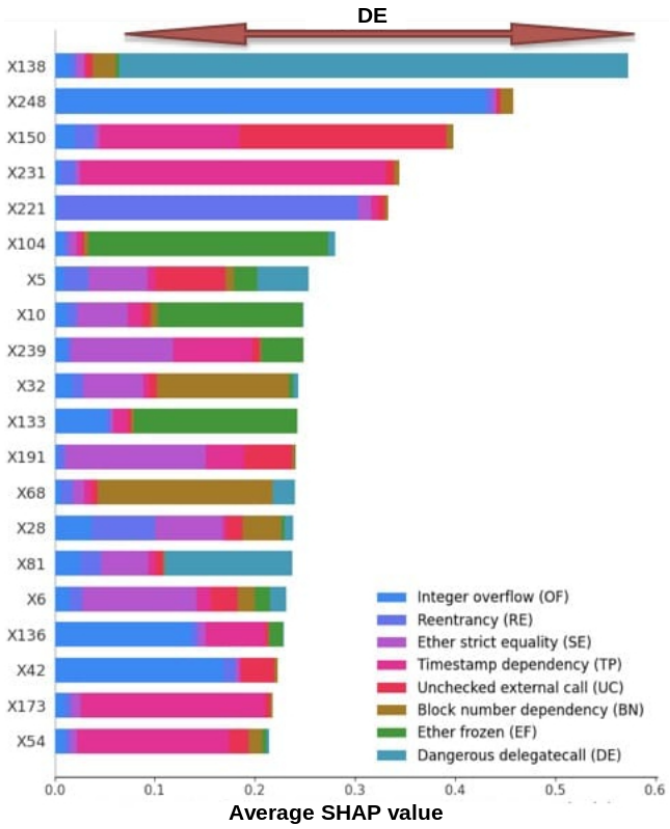

Fig. 3. SHAP Feature Impact

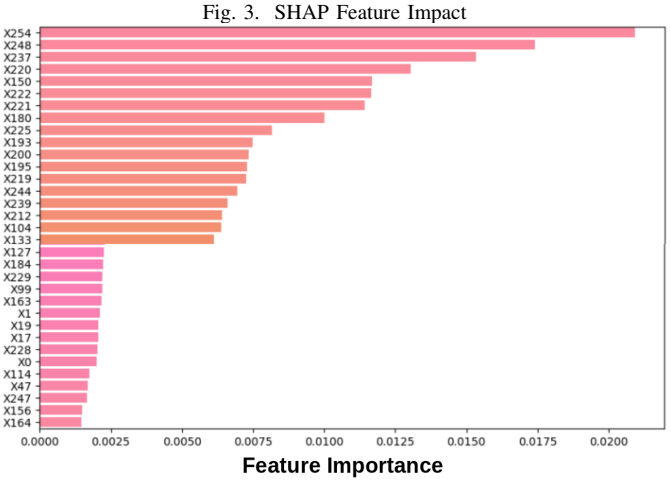

Fig. 4. XGB Feature Importance

average SHAP value between 0.096 and 0.58, equating to a magnitude of 0.484. However, the same feature has a lesser impact on "Timestamp dependency," ranging from 0.05 to 0.056, a magnitude of 0.06. For the "Integer Overflow (IO)" vulnerability, the feature has a minimal effect, with an average SHAP value between 0.0 and 0.03, equating to a magnitude of 0.03.

Likewise, for the "Ether Strict Equality (SE)" vulnerability, the feature has a modest impact, with an average SHAP value of 0.03 to 0.05, a magnitude of 0.02. The feature also has a low

impact on the ”Block Number Dependency (BN)” vulnerability, with a magnitude of 0.04. Thus, the feature influences various vulnerabilities, albeit at different scales. This allows XAI to make ML model outcomes interpretable. Figures 3 and 4 indicate that feature X138 has the highest SHAP value of 0.58 magnitude, while in XGBClassifier, it registers a lower value of 0.0055, signaling lesser relevance. Furthermore, the feature importance in Figure 4 doesn't correlate with any specific smart contract vulnerabilities. In XGBClassifier, feature X164 has the least relevance (around 0.055 magnitudes), whereas feature X254 has the maximum value. Features not shown in Figure 3 are not relevant in the SHAP global explanation for identifying smart contract vulnerabilities, whereas the XGBClassifier model assigns varying degrees of importance to most features.

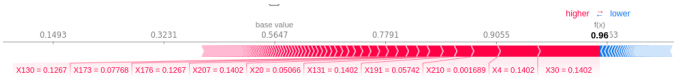

Fig. 5. SHAP Feature explanation 1

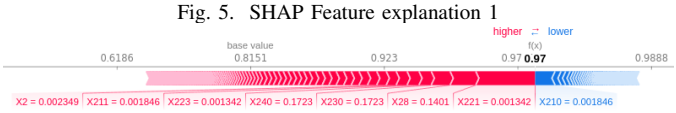

Fig. 6. SHAP Feature explanation 2

The system outlined in this study was employed to identify vulnerabilities in two external smart contracts not present in the training and testing dataset. Figure 5 depicts the SHAP model's local explanation for the first of these external smart contracts, as it pertains to a specific input. The figure also demonstrates how SHAP assesses the likelihood of encountering a vulnerability class named 'Ether strict equality' in association with this first external smart contract. In this case, the probability stands at 96%. Features in the red-marked area elevate this probability, whereas those in the blue-marked area reduce it. From this particular smart contract data, it's evident that features X30, X4, and X210 have the most significant impact, each with a 96% probability.

Figure 6 shows the explanation generated for the second external smart contract, identifying the vulnerability class as 'Reentrancy'. The explanation reveals that the likelihood of this Reentrancy vulnerability is 97%. Here, features X221, X28, and X230 exert a more significant impact compared to X210 and others. Thus, the system crafted in this study is versatile enough to detect and elucidate vulnerabilities in any smart contract.

An example explanation using the LIME model is displayed in Figure 7.

According to the LIME model, the likelihood of the Ether strict equality vulnerability stands at 89%. Nearly all features, except for X150, X190, X209, X180, X248, X213, X191, X221, X222, and X113, contribute to this 89% probability for Ether strict equality vulnerability. In this test instance, features X150, X190, X209, X248, X191, and X222 influence the Unchecked external call vulnerability, while features X180,

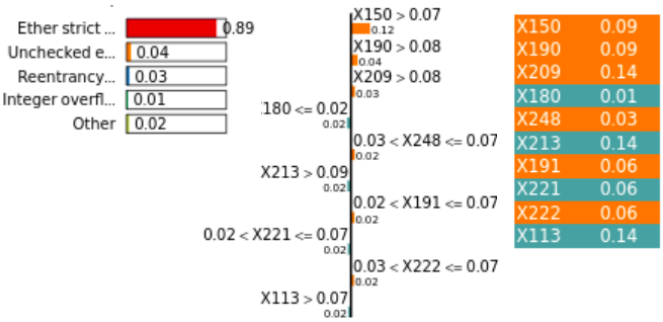

Fig. 7. LIME explanation

X213, X221, and X113 affect the Integer overflow vulnerability.

Tables III and IV display the evolutionary matrices for SHAP and LIME when used with XGBoost and ANN classifiers. Notably, given their integration with SHAP and LIME, these classifiers can also be called XGBoostSHAP, XGBoostLIME, ANNSHAP, and ANNLIME. As the performance of SHAP and LIME is contingent on their base classifiers, which are XGBoost and ANN in this study, the accuracy levels are comparable: 97% for XGBoost and 94% for ANN. SHAP outperforms LIME in terms of feature coverage and relevance for both XGBoost and ANN classifiers. Conversely, LIME exhibits fewer consistency errors in both classifiers. The larger consistency error in SHAP arises because it accounts for all possible feature combinations and their impacts on a prediction, whereas LIME focuses on individual features when making a prediction, resulting in lower consistency error. Overall, SHAP is more effective as it offers higher coverage and relevancy values.

TABLE III
MODEL EVALUATION METRICS FOR SHAP

Model	Accuracy	F. Coverage	Relevance	Consistency Error
XGBoost	0.97	0.86	0.02	0.034
ANN	0.94	0.87	0.90	0.013

TABLE IV
MODEL EVALUATION METRICS FOR LIME

Model	Accuracy	F. Coverage	Relevance	Consistency Error
XGBoost	0.97	0.039	0.001	0.002
ANN	0.94	0.039	0.002	0.003

F. Segmentation

The tables V and VI present a series of instructions for the Ethereum Virtual Machine, focusing on features X28 and X221 from the 2nd external smart contract data. These features' corresponding tokens are detokenized, as outlined in the methodology section. By matching these with the CFG and RBW generated during preprocessing, the corresponding opcodes are identified. Each row in the table represents a unique opcode, followed by a designation that describes

TABLE V
CORRESPONDING OPCODE FOR X221 FEATURE FROM EXTERNAL DATA2

Opcode	Name	Operand 1	Operand 2	Operand 3	Gas	Description	Jumpdest	Program counter
0x5b	JUMPDEST	0	0	0	1	Mark a valid destination for jumps.	None	203
0x61	PUSH	2	0	1	3	Place 2-byte item on stack.	211	204
0x61	PUSH	2	0	1	3	Place 2-byte item on stack.	944	207
0x56	JUMP	0	1	0	8	Alter the program counter.	None	210

TABLE VI
CORRESPONDING OPCODE FOR X28 FEATURE FROM EXTERNAL DATA2

Opcode	Name	Operand 1	Operand 2	Operand 3	Gas	Description	Jumpdest	PC
0x5b	JUMPDEST	0	0	0	1	Mark a valid destination for jumps.	None	177
0x61	PUSH	2	0	1	3	Place 2-byte item on stack.	221	178
0x60	PUSH	1	0	1	3	Place 1 byte item on stack.	4	181
0x80	DUP	0	1	2	3	Duplicate 1st stack item.	None	183
0x80	DUP	0	1	2	3	Duplicate 1st stack item.	None	184
0x35	CALLDATALOAD	0	1	1	3	Get input data of current environment.	None	185
0x73	PUSH	20	0	1	3	Place 20-byte item on stack.	1461501637	186
0x16	AND	0	2	1	3	Bitwise AND operation.	None	207
0x90	SWAP	0	2	2	3	Exchange 1st and 2nd stack items.	None	208
0x60	PUSH	1	0	1	3	Place 1 byte item on stack.	32	209
0x1	ADD	0	2	1	3	Addition operation.	None	211
0x90	SWAP	0	2	2	3	Exchange 1st and 2nd stack items.	None	212
0x91	SWAP	0	3	3	3	Exchange 1st and 3rd stack items.	None	213
0x90	SWAP	0	2	2	3	Exchange 1st and 2nd stack items.	None	214
0x50	POP	0	1	0	2	Remove item from stack.	None	215
0x50	POP	0	1	0	2	Remove item from stack.	None	216
0x61	PUSH	2	0	1	3	Place 2-byte item on stack	260	217
0x56	JUMP	0	1	0	8	Alter the program counter	None	220

the operation and up to three operands providing additional details about the operation. The gas cost for each operation indicates the computational effort required for its execution. The "description" column offers a brief explanation of each operation's functionality. Some operations have a 'jumpdest' column, signifying that they can serve as jump destinations. The PC column indicates the operation's position in the program counter.

Taken together, these columns furnish a detailed outline of how a transaction or smart contract should be executed on the Ethereum blockchain. Notably, the presence of the JUMP instruction (0x56) in this code segment could enable a reentrancy attack within the EVM, as it can alter the program counter and divert the control flow to another part of the contract. If an attacker manages to trigger this code section repeatedly before the initial execution finishes, they could exploit this behavior to continually call the same function, potentially draining the contract of its financial resources or carrying out other malicious activities. Moreover, the code contains several PUSH instructions for loading user-provided input data onto the stack. Lack of adequate input validation could make the contract vulnerable to exploitation. To minimize the risk of reentrancy attacks, it is advisable to implement a safeguard that prevents a function from being invoked again until the previous execution has completed.

V. CONCLUSION

This paper outlines the methodology used for identifying vulnerabilities in smart contracts, as well as the associated code segments and causative features. We utilized a range of Machine Learning and Deep Learning algorithms, with the highest accuracy seen in Extreme Gradient Boosting, Hist Gradient Boosting, Bagging, Decision Trees, K-Nearest Neighbors, and 1D Convolutional Neural Networks. The data preprocessing involved multiple steps like bytecode generation, opcode conversion, CFG construction, and feature selection through tokenization and TFIDF application. By incorporating CFG into preprocessing, we effectively blended Graph-based techniques with ML and CNN approaches, yielding a holistic insight into smart contract vulnerabilities. We also employed SHAP and LIME as Explainable AI models to determine the features triggering vulnerabilities in smart contracts, facilitating a deeper understanding and interpretation of ML-generated outcomes. Metrics like accuracy, coverage, and relevancy were used to gauge the efficacy of these XAI models, with SHAP emerging as the more proficient tool for interpreting vulnerabilities in smart contracts. As demonstrated in Tables V and VI, we've identified code sections corresponding to specific vulnerabilities, aiding in corrective actions to mitigate hacking risks. Going forward, the focus will be on fine-tuning the balance between model accuracy and explainability, a vital aspect for advancing effective XAI models.

REFERENCES

[1] Al-E'mari, S., Anbar, M., Sanjalawe, Y., Manickam, S.: A labeled transactions-based dataset on the ethereum network. In: International Conference on Advances in Cyber Security. pp. 61–79. Springer (2020)

[2] Christian, H., Agus, M.P., Suhartono, D.: Single document automatic text summarization using term frequency-inverse document frequency (tf-idf). ComTech: Computer, Mathematics and Engineering Applications **7**(4), 285–294 (2016)

[3] Došilović, F.K., Brčić, M., Hlupić, N.: Explainable artificial intelligence: A survey. In: 2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO). pp. 0210–0215. IEEE (2018)

[4] Fior, J., Cagliero, L., Garza, P.: Leveraging explainable ai to support cryptocurrency investors. Future Internet **14**(9), 251 (2022)

[5] Guo, H., Yu, X.: A survey on blockchain technology and its security. Blockchain: research and applications **3**(2), 100067 (2022)

[6] Holzinger, A., Saranti, A., Molnar, C., Biecek, P., Samek, W.: Explainable ai methods-a brief overview. In: International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers. pp. 13–38. Springer (2020)

[7] Huang, J., Zhou, K., Xiong, A., Li, D.: Smart contract vulnerability detection model based on multi-task learning. Sensors (Basel, Switzerland) **22** (2022)

[8] Hwang, S., Choi, S.H., Shin, J., Choi, Y.H.: Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection. IEEE Access **10**, 32595–32607 (2022)

[9] Jiang, B., Liu, Y., Chan, W.K.: Contractfuzzer: Fuzzing smart contracts for vulnerability detection. 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE) pp. 259–269 (2018)

[10] Khan, S.N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., Bani-Hani, A.: Blockchain smart contracts: Applications, challenges, and future trends. Peer-to-peer Networking and Applications **14**, 2901–2925 (2021)

[11] Li, N., Liu, Y., Li, L., Wang, Y.Y.: Smart contract vulnerability detection based on deep and cross network. 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA) pp. 533–536 (2022)

[12] Linardatos, P., Papastefanopoulos, V., Kotsiantis, S.: Explainable ai: A review of machine learning interpretability methods. Entropy **23**(1), 18 (2020)

[13] Liu, Z., Qian, P., Wang, X., Zhu, L., He, Q., Ji, S.: Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion. ArXiv **abs/2106.09282** (2021)

[14] Liu, Z., Qian, P., Yang, J., Liu, L., Xu, X., He, Q., Zhang, X.: Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting. arXiv preprint arXiv:2301.03943 (2023)

[15] Lorenz, J., Silva, M.I., Aparício, D., Ascensão, J.T., Bizarro, P.: Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity. In: Proceedings of the first ACM international conference on AI in finance. pp. 1–8 (2020)

[16] Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. Advances in neural information processing systems **30** (2017)

[17] Mehar, M.I., Shier, C.L., Giambattista, A., Gong, E., Fletcher, G., Sanayhie, R., Kim, H.M., Laskowski, M.: Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. Journal of Cases on Information Technology (JCIT) **21**(1), 19–32 (2019)

[18] Qian, P., Liu, Z., He, Q., Huang, B., Tian, D., Wang, X.: Smart contract vulnerability detection technique: A survey. ArXiv **abs/2209.05872** (2022)

[19] Ribeiro, M.T., Singh, S., Guestrin, C.: " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1135–1144 (2016)

[20] Sun, Y., Gu, L.: Attention-based machine learning model for smart contract vulnerability detection. Journal of Physics: Conference Series **1820** (2021)

[21] Wang, B., Chu, H., Zhang, P., Dong, H.: Smart contract vulnerability detection using code representation fusion. 2021 28th Asia-Pacific Software Engineering Conference (APSEC) pp. 564–565 (2021)

[22] Wu, H., Zhang, Z., Wang, S., Lei, Y., Lin, B., Qin, Y., Zhang, H., Mao, X.: Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques. 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE) pp. 378–389 (2021)

[23] Wu, H., Dong, H., He, Y., Duan, Q.: Smart contract vulnerability detection based on hybrid attention mechanism model. Applied Sciences (2023)

[24] Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., Zhu, J.: Explainable ai: A brief survey on history, research areas, approaches and challenges. In: Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8. pp. 563–574. Springer (2019)

[25] Zhang, X., Li, J., Wang, X.: Smart contract vulnerability detection method based on bi-lstm neural network. 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA) pp. 38–41 (2022)