Leakage-Safe Proxy Smart Contract Detection Using Solidity Structural Graphs and Graph Neural Networks for Secure Metaverse Blockchain Applications

S. Aasha Nandhini[1,*], Karthick Manoj[2]

[1]Department of Electronics and Communication Engineering, Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603110, India

[2]Department of Electrical and Electronics Engineering, AMET University, Chennai, india

ABSTRACT

Upgradeable proxy smart contracts have become a fundamental component of modern blockchain systems because they enable contract functionality to evolve without changing deployed contract addresses. Their widespread adoption in decentralized applications and Metaverse platforms introduces additional architectural complexity that complicates security analysis and automated auditing. This study proposes a Graph Neural Network (GNN)-based framework for proxy smart contract detection using a Solidity structural graph representation combined with a leakage-safe evaluation strategy. Solidity source code is transformed into graph structures that preserve syntactic and structural relationships among program elements, allowing GNN models to learn architectural patterns associated with proxy contracts. Three GNN architectures, namely Graph Convolutional Network (GCN), Graph Attention Network (GAT), and GraphSAGE, are evaluated using leakage-safe grouped cross-validation to eliminate information leakage caused by duplicated source code. Experiments were conducted on a dataset containing 377 Solidity smart contracts, which were grouped into 146 unique source-code groups after duplicate removal. The experimental results demonstrate that GCN achieved the highest ranking performance with a ROC-AUC of 0.851 and a PR-AUC of 0.606, while GraphSAGE obtained the highest classification performance with an F1-score of 0.279, a Matthews Correlation Coefficient of 0.312, and a recall of 0.500. The leakage audit confirmed that no duplicated source-code groups were shared across training, validation, and testing partitions, ensuring reliable model evaluation. These findings demonstrate that structural graph representations combined with leakage-safe evaluation provide an effective and reliable approach for proxy smart contract detection, contributing to blockchain security and supporting secure smart contract auditing in Metaverse applications.

Keywords Graph Neural Network, Proxy Smart Contract Detection, Blockchain Security, Solidity, Metaverse

Submitted: 25 April 2026
Accepted: 15 July 2026
Published: 17 November 2026

Corresponding author
S. Aasha Nandhini,
aashanandhinis@ssn.edu.in

Additional Information and Declarations can be found on page 326

DOI: 10.47738/ijrm.v3i4.89

Copyright
2026 Nandhini and Manoj

Distributed under
Creative Commons CC-BY 4.0

INTRODUCTION

Blockchain technology has become one of the fundamental infrastructures supporting decentralized applications by enabling transparent, immutable, and trustless transaction processing without relying on centralized authorities. Beyond cryptocurrency, blockchain has been widely adopted in decentralized finance, digital identity management, supply chain systems, healthcare, and Internet of Things applications. More recently, blockchain has also become a core component of Metaverse ecosystems, where it facilitates digital asset

ownership, virtual land management, decentralized marketplaces, governance mechanisms, and non-fungible token (NFT) transactions [1]. Smart contracts serve as the programmable foundation of these blockchain-based services by automatically executing predefined rules and agreements. As Metaverse platforms continue to evolve, the security and reliability of smart contracts become increasingly important because vulnerabilities or architectural weaknesses may directly affect digital assets and user trust.

To support continuous software evolution while preserving deployed contract addresses, many blockchain applications adopt upgradeable proxy smart contracts. Instead of embedding all business logic within a single contract, proxy contracts delegate function execution to separate implementation contracts through delegation mechanisms. This architecture enables developers to upgrade contract functionality without migrating user data or modifying existing blockchain addresses. Although upgradeability significantly improves maintainability and system flexibility, it also introduces additional architectural complexity that complicates security auditing. Conventional smart contract analysis tools often assume that the analyzed contract directly contains the executable business logic, whereas proxy contracts distribute their functionality across multiple interacting contracts. Consequently, the ability to automatically identify proxy contracts before vulnerability assessment has become an essential prerequisite for blockchain security analysis [2].

Recent advances in machine learning have introduced automated approaches for smart contract analysis, including vulnerability detection, code classification, malware identification, and contract categorization. Earlier studies primarily relied on handcrafted software metrics, opcode frequency, bytecode features, or token-based representations extracted from Solidity source code. While these approaches have demonstrated promising performance, they generally fail to preserve the structural relationships among program elements. Since smart contracts are naturally represented as interconnected structures through functions, inheritance hierarchies, control flow, and syntax trees, graph-based representations have attracted increasing attention. Graph Neural Networks (GNNs), including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), and GraphSAGE, have shown superior capability in learning structural dependencies by propagating information through graph topology. These characteristics make GNNs particularly suitable for analyzing proxy smart contracts, whose distinguishing features are primarily architectural rather than lexical [3].

Despite these advances, several limitations remain in existing studies. First, most machine learning approaches for smart contract analysis focus on vulnerability detection, malicious contract identification, or code clone detection, whereas proxy smart contract detection has received considerably less attention. Second, previous studies frequently employ token sequences, opcode embeddings, or bytecode representations that may overlook higher-level structural relationships present in Solidity source code. Third, although duplicated smart contracts are common due to template reuse, inheritance libraries, and repeated deployment of standardized implementations, many existing studies rely on conventional random train-test splitting without explicitly preventing source-code duplication across data partitions. Such evaluation strategies may introduce information leakage, allowing identical or nearly identical contracts to appear in both training and testing datasets, thereby

producing overly optimistic performance estimates. Consequently, the true generalization capability of machine learning models remains difficult to assess [4].

To address these limitations, this study proposes a Graph Neural Network-based proxy smart contract detection framework using a Solidity structural graph representation combined with a leakage-safe evaluation strategy. Solidity source code is transformed into graph structures that preserve syntactic and structural relationships among program elements, enabling Graph Neural Networks to learn architectural characteristics associated with proxy contracts. Three representative GNN architectures, namely Graph Convolutional Network (GCN), Graph Attention Network (GAT), and GraphSAGE, are comparatively evaluated using leakage-safe grouped cross-validation, where contracts sharing identical source code are assigned exclusively to the same evaluation partition. This strategy prevents information leakage and provides a more reliable assessment of model generalization.

The experimental evaluation demonstrates that the proposed framework effectively distinguishes proxy and non-proxy smart contracts under severe class imbalance while maintaining leakage-free evaluation. The results show that GCN achieved the highest-ranking performance, whereas GraphSAGE provided the best balance between precision and recall. More importantly, the proposed leakage-safe evaluation framework ensures that the reported performance reflects genuine generalization rather than memorization of duplicated contracts. Therefore, this study contributes not only a graph-based proxy smart contract detection approach but also a rigorous evaluation methodology that improves the reliability of machine learning research for blockchain security. These contributions are particularly relevant to Metaverse applications, where upgradeable smart contracts play a central role in supporting secure, maintainable, and continuously evolving decentralized services [5].

Literature Review and Related Works

Blockchain technology has become a fundamental infrastructure for decentralized applications by enabling transparent, immutable, and trustless execution of digital transactions through smart contracts. The increasing adoption of blockchain in decentralized finance, digital asset management, supply chain systems, and Metaverse platforms has significantly expanded the role of smart contracts in automating business processes without centralized control. However, the growing complexity of smart contract applications has also increased the number of security risks, making automated security analysis an important research topic. Recent surveys have emphasized that smart contract vulnerabilities remain one of the primary threats affecting blockchain reliability and decentralized ecosystems [6].

Early smart contract security analysis primarily relied on static analysis, symbolic execution, formal verification, and rule-based detection techniques. These approaches are capable of identifying predefined vulnerability patterns with relatively high interpretability. Nevertheless, their effectiveness depends heavily on manually designed rules and expert knowledge, making them difficult to adapt to newly emerging attack patterns or increasingly complex smart contract architectures. Furthermore, symbolic execution and formal verification often suffer from scalability issues when applied to large-scale blockchain

systems [7].

To overcome these limitations, machine learning approaches have been introduced for automated smart contract analysis. Early learning-based methods represented smart contracts using opcode sequences, bytecode embeddings, software metrics, or tokenized Solidity source code. Although these representations achieved promising classification performance, they generally failed to preserve structural relationships among program components, such as inheritance hierarchies, function interactions, and control flow dependencies. As a result, important semantic information embedded in smart contract structures could not be fully exploited during feature learning [8].

Graph-based program representation has recently emerged as an effective alternative for modeling source code because graphs naturally preserve structural relationships among program elements. Instead of treating source code as a sequential token stream, graph representations encode syntactic and semantic dependencies through interconnected nodes and edges. This representation enables machine learning models to capture complex interactions between functions, statements, and program structures that are difficult to represent using conventional sequence-based approaches [9].

The development of Graph Neural Networks has further improved the capability of graph-based program analysis. Graph Convolutional Networks (GCN) perform feature aggregation by propagating information through neighboring nodes, allowing structural information to be learned across the entire graph. This mechanism has demonstrated superior performance for various software engineering tasks, including vulnerability detection and source code classification, because it effectively captures contextual information contained within graph topology [10].

To further improve graph representation learning, Graph Attention Networks (GAT) introduce attention mechanisms that assign different importance weights to neighboring nodes during feature aggregation. By learning adaptive neighborhood weights, GAT can emphasize structurally important program components while suppressing less informative connections. This attention mechanism has shown promising results for graph classification problems involving heterogeneous structural information [11].

Another widely adopted Graph Neural Network architecture is GraphSAGE, which performs neighborhood sampling instead of aggregating information from the entire graph. This sampling strategy improves computational efficiency and enables scalable representation learning on larger graphs. In software engineering applications, GraphSAGE has demonstrated competitive performance by preserving local structural information while reducing computational complexity during graph embedding generation [12].

Recent studies have successfully applied Graph Neural Networks to smart contract vulnerability detection by transforming Solidity programs into Abstract Syntax Trees, Control Flow Graphs, or hybrid graph representations. These graph-based approaches consistently outperform traditional token-based methods because they preserve richer structural information that reflects program behavior more accurately. Experimental results indicate that Graph Neural Networks are capable of learning complex semantic relationships that are difficult to capture using handcrafted software metrics [13].

Despite these advances, most existing studies focus primarily on vulnerability

detection, including reentrancy attacks, integer overflow, timestamp dependence, and access control vulnerabilities. Comparatively little attention has been given to proxy smart contract detection, despite the widespread adoption of upgradeable proxy contracts in Ethereum-based decentralized applications. Existing proxy identification techniques are generally based on handcrafted rules or static signature matching, limiting their ability to generalize across different proxy implementations [14].

Another important issue concerns the evaluation methodology adopted by previous machine learning studies. Public smart contract repositories frequently contain duplicated contracts because developers commonly reuse standard templates, inheritance libraries, and previously deployed implementations. However, many existing studies continue to evaluate models using conventional random train-test splitting without explicitly preventing duplicated source code from appearing across different data partitions. Such evaluation procedures may introduce information leakage and produce overly optimistic performance estimates that do not accurately reflect model generalization [15].

Several recent studies have emphasized the importance of designing more reliable evaluation protocols for machine learning models applied to source code analysis. Group-aware validation strategies have been shown to produce more realistic performance estimates by ensuring that duplicated or highly similar source-code instances remain within the same evaluation partition. These findings indicate that leakage prevention should become an essential component of future smart contract learning frameworks [16].

The rapid expansion of blockchain technology within Metaverse applications further increases the importance of reliable smart contract analysis. Digital asset ownership, virtual land management, decentralized governance, and blockchain-based marketplaces increasingly depend on upgradeable smart contracts to support continuous system evolution. Consequently, accurate identification of proxy contracts has become an important prerequisite for subsequent vulnerability assessment and security auditing in Metaverse ecosystems [17].

Although Graph Neural Networks have demonstrated strong performance for graph-based software analysis, comparative studies involving multiple GNN architectures under leakage-safe evaluation remain limited. In particular, comprehensive comparisons among GCN, GAT, and GraphSAGE for proxy smart contract detection have not been sufficiently investigated. Furthermore, structural graph representations specifically designed for Solidity source code remain relatively unexplored [18].

Based on these observations, this study proposes a Graph Neural Network framework for proxy smart contract detection using a Solidity structural graph representation combined with leakage-safe grouped cross-validation. The proposed framework aims to preserve structural program information while ensuring that duplicated source-code groups never appear across different evaluation partitions. By integrating structural graph learning with rigorous leakage prevention, this study seeks to provide a more reliable and practical solution for blockchain security analysis in modern decentralized and Metaverse applications [19], [20].

Methodology

Research Framework

The overall research procedure is presented in figure 1. As shown in the figure, the proposed proxy smart contract detection framework consists of six main stages: dataset preparation, Solidity source-code preprocessing, duplicate-group identification, structural graph construction, Graph Neural Network training, and leakage-safe performance evaluation. First, Solidity smart contracts are collected and labeled as proxy or non-proxy contracts. The source code is then normalized by removing comments and standardizing whitespace. A SHA-256 hash is calculated from each normalized contract to identify identical source-code groups. Subsequently, every contract is converted into a structural graph whose nodes represent Solidity statements and whose edges preserve sequential, hierarchical block, and approximate internal function-call relationships. Each graph node is represented using a 48-dimensional feature vector. The resulting graphs are used to train three architectures, namely Graph Convolutional Network, Graph Attention Network, and GraphSAGE. Finally, the models are evaluated using grouped cross-validation in which identical source-code groups are kept in the same partition to prevent information leakage.

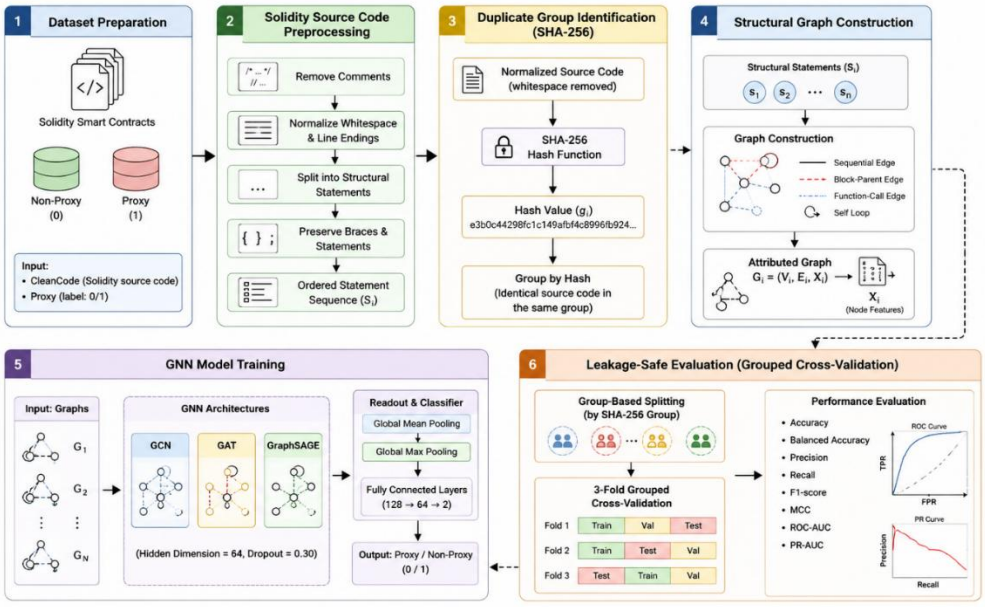


Figure 1 Research steps of the proposed proxy smart contract detection framework.

Dataset Preparation and Label Representation

The dataset contains Solidity smart contracts classified into two categories, namely proxy and non-proxy contracts. Each contract C_i is associated with a binary class label y_i, defined as:

$$y_i = \begin{cases} 1, & \text{if } C_i \text{ is a proxy contract,} \\ 0, & \text{if } C_i \text{ is a non-proxy contract.} \end{cases} \quad (1)$$

The code implementation expects the Solidity source code in the CleanCode field and the corresponding class label in the Proxy field. Before graph generation, each contract is normalized to reduce irrelevant formatting variations while preserving its structural content.

Solidity Source-Code Preprocessing

Raw Solidity source code may contain comments, inconsistent line endings, repeated whitespace, and other formatting differences that do not change program behavior. The preprocessing stage first removes both multiline and single-line comments. Let $C_i$denote the original source code and let $\mathcal{N}(\cdot)$denote the normalization operation. The normalized contract is expressed as:

$$\tilde{C}_i = \mathcal{N}(C_i), \tag{2}$$

$\mathcal{N}(\cdot)$includes comment removal, line-ending normalization, whitespace standardization, and removal of unnecessary blank lines.

The normalized code is subsequently segmented into lightweight structural statements using braces and semicolons as boundaries. Semicolons are used only as delimiters, whereas opening and closing braces are retained as separate structural statements. Therefore, the processed contract can be represented as an ordered sequence

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{in_i}\}, \tag{3}$$

$s_{ij}$is the j-th structural statement and $n_i$is the total number of statements in contract C_i. When no valid statement is produced, the implementation creates one placeholder node named EMPTY_CONTRACT to maintain a valid graph structure.

Duplicate Identification Using SHA-256

Smart contract datasets commonly contain identical implementations resulting from contract templates, copied libraries, or repeated deployments. To detect exact duplicates, all whitespace is removed from the normalized source code before a SHA-256 digest is calculated. For contract C_i, its group identifier is defined as:

$$g_i = \mathrm{SHA256}\left(\mathrm{RemoveWhitespace}(\tilde{C}_i)\right). \tag{4}$$

Two contracts are assigned to the same source-code group when their hash values are identical:

$$C_i \sim C_j \iff g_i = g_j. \tag{5}$$

This grouping mechanism is later used during data splitting. It ensures that contracts with identical normalized source code cannot be distributed across training, validation, and testing subsets.

Solidity Structural Graph Construction

Each preprocessed contract is transformed into an attributed graph

$$G_i = (V_i, E_i, X_i, y_i), \tag{6}$$

$V_i$is the set of structural statement nodes, $E_i$is the set of graph edges, $X_i$is the node-feature matrix, and $y_i$is the contract-level class label.

For a contract containing $n_i$structural statements, each statement is

represented as one graph node:

$$V_i = \{v_{i1}, v_{i2}, \ldots, v_{in_i}\}. \quad (7)$$

The edge set combines four relationship types:

$$E_i = E_i^{\text{seq}} \cup E_i^{\text{block}} \cup E_i^{\text{call}} \cup E_i^{\text{self}}. \quad (8)$$

Sequential edges connect consecutive statements to preserve source-code order. For each adjacent node pair, bidirectional edges are added:

$$E_i^{\text{seq}} = \{(v_{ij}, v_{i,j+1}), (v_{i,j+1}, v_{ij}) \mid 1 \le j < n_i\}. \quad (9)$$

A stack-based procedure is used to track opening and closing braces. Statements located inside a block are connected to the node representing the corresponding opening brace. The opening brace is also linked to its matching closing brace. These edges are inserted in both directions:

$$E_i^{\text{block}} = \{(v_p, v_c), (v_c, v_p) \mid v_c \text{ belongs to the block represented by } v_p\}. \quad (10)$$

This relationship preserves hierarchical structures such as contracts, functions, conditional branches, and loop bodies.

Function declarations are identified using Solidity function-name patterns. When a statement contains a call to a declared function, the call node is connected to the corresponding declaration node:

$$E_i^{\text{call}} = \{(v_{ij}, v_{ik}), (v_{ik}, v_{ij}) \mid s_{ij} \text{ calls the function declared in } s_{ik}\}. \quad (11)$$

These edges provide an approximate representation of internal call relationships. They are based on source-code pattern matching rather than a complete semantic call-graph analysis.

A self-loop is added to every graph node:

$$E_i^{\text{self}} = \{(v_{ij}, v_{ij}) \mid 1 \le j \le n_i\}. \quad (12)$$

elf-loops guarantee valid message propagation for isolated and single-node graphs.

Graph-based representations have been shown to preserve syntactic and semantic relationships that may be lost in purely sequential source-code representations 16.

Node Feature Representation

Each structural node is described by a 48-dimensional feature vector:

$$\mathbf{x}_{ij} = \left[x_{ij}^{(1)}, x_{ij}^{(2)}, \ldots, x_{ij}^{(48)}\right]^{\top} \in \mathbb{R}^{48}. \quad (13)$$

The complete node-feature matrix of graph $G_i$is

$$X_i = \begin{bmatrix} \mathbf{x}_{i1}^{\top} \\ \mathbf{x}_{i2}^{\top} \\ \vdots \\ \mathbf{x}_{in_i}^{\top} \end{bmatrix} \in \mathbb{R}^{n_i \times 48}. \quad (14)$$

The feature vector combines binary lexical indicators and normalized numerical characteristics. The binary indicators represent Solidity constructs such as functions, constructors, modifiers, fallback and receive functions, delegatecall, low-level calls, staticcall, callcode, selfdestruct, require, assert, revert, control-flow statements, events, mappings, data-location keywords, visibility modifiers, data types, assembly blocks, and proxy-related terms such as proxy, implementation, upgrade, admin, and slot.

For a keyword pattern k, the corresponding binary indicator is defined as:

$$x_{ij}^{(k)} = \begin{cases} 1, & \text{if pattern } k \text{ occurs in } s_{ij}, \\ 0, & \text{otherwise.} \end{cases} \quad (15)$$

The numerical features are normalized using predefined upper bounds. For example, statement length is represented as:

$$x_{ij}^{(\text{length})} = \frac{\min\left(| s_{ij} |, 500\right)}{500}, \quad (16)$$

$| s_{ij} |$denotes the number of characters in the statement.

The parenthesis count is calculated as:

$$x_{ij}^{(\text{parenthesis})} = \frac{\min\left(N_{(}(s_{ij}) + N_{)}(s_{ij}), 20\right)}{20}, \quad (17)$$

while the bracket count is:

$$x_{ij}^{(\text{bracket})} = \frac{\min\left(N_{[}(s_{ij}) + N_{]}(s_{ij}), 20\right)}{20}. \quad (18)$$

The normalized brace depth is defined as:

$$x_{ij}^{(\text{depth})} = \frac{\min\left(\max(d_{ij}, 0), 20\right)}{20}, \quad (19)$$

d_{ij}denotes the nesting depth of statement s_{ij}. Assignment, logical, and arithmetic operator counts are normalized in a similar manner using their corresponding thresholds.

Graph Neural Network Architectures

Three Graph Neural Network architectures are evaluated using the same graph representation, hidden dimension, pooling mechanism, and classifier. Each architecture contains three graph-processing layers. Rectified Linear Unit activation is applied after every layer, while dropout is applied after the first and second layers. Previous smart contract studies have also demonstrated the

value of graph convolution and attention mechanisms for learning structural code representations 17.

For the GCN model, the propagation rule at layer $l+1$can be expressed as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{20}$$

$H^{(l)}$is the node-representation matrix at layer l, $\tilde{A}$is the adjacency matrix including self-loops, $\tilde{D}$is its degree matrix, $W^{(l)}$is the learnable weight matrix, and $\sigma(\cdot)$is the ReLU activation function.

The initial node representation is:

$$H^{(0)} = X_i. \tag{21}$$

Three GCN layers are used, with the hidden dimension set to 64.

The GAT model computes attention coefficients to determine the relative importance of neighboring nodes. For connected nodes $v_i$and v_j, the unnormalized attention score is:

$$e_{ij} = \mathrm{LeakyReLU}\left(\mathbf{a}^{\top}\left[W\mathbf{h}_i \parallel W\mathbf{h}_j\right]\right), \tag{22}$$

$W$is a learnable transformation matrix, $\mathbf{a}$is the attention parameter, and $\parallel$denotes concatenation.

The normalized coefficient is:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)}\exp(e_{ik})}. \tag{23}$$

The updated representation of node $i$is:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\, W\mathbf{h}_j^{(l)}\right). \tag{24}$$

In the implementation, the first two GAT layers use four attention heads. The last GAT layer uses one attention head and produces a 64-dimensional node representation.

GraphSAGE updates each node representation by aggregating the features of its neighboring nodes. A general mean-aggregation formulation is:

$$\mathbf{m}_i^{(l)} = \frac{1}{|\,\mathcal{N}(i)\,|}\sum_{j\in\mathcal{N}(i)}\mathbf{h}_j^{(l)}, \tag{25}$$

followed by:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(W^{(l)}\left[\mathbf{h}_i^{(l)} \parallel \mathbf{m}_i^{(l)}\right]\right). \tag{26}$$

The GraphSAGE implementation also consists of three layers with a hidden dimension of 64.

Graph-Level Pooling and Classification

After the third graph layer, node representations are aggregated into one graph-level vector using both global mean pooling and global max pooling. Mean pooling is defined as:

$$\mathbf{h}_i^{\text{mean}} = \frac{1}{|V_i|} \sum_{v \in V_i} \mathbf{h}_v, \tag{27}$$

max pooling is:

$$\mathbf{h}_i^{\max} = \max_{v \in V_i} \mathbf{h}_v. \tag{28}$$

The final graph embedding is obtained through concatenation:

$$\mathbf{z}_i = \mathbf{h}_i^{\text{mean}} \parallel \mathbf{h}_i^{\max}. \tag{29}$$

Because each pooling vector has 64 dimensions, $\mathbf{z}_i$has 128 dimensions. It is passed through a fully connected classifier:

$$\mathbf{r}_i = \text{ReLU}(W_1 \mathbf{z}_i + \mathbf{b}_1), \tag{30}$$

$$\mathbf{o}_i = W_2 \text{Dropout}(\mathbf{r}_i) + \mathbf{b}_2, \tag{31}$$

$\mathbf{o}_i \in \mathbb{R}^2$contains the logits for the non-proxy and proxy classes.

The proxy-class probability is computed using softmax:

$$p_i = P(y_i = 1 \mid G_i) = \frac{\exp(o_{i1})}{\exp(o_{i0}) + \exp(o_{i1})}. \tag{32}$$

The predicted label is

$$\hat{y}_i = \begin{cases} 1, & p_i \geq 0.5, \\ 0, & p_i < 0.5. \end{cases} \tag{33}$$

Graph-level pooling has been widely used to convert variable-sized code graphs into fixed-dimensional representations suitable for classification 18.

Leakage-Safe Grouped Cross-Validation

The evaluation procedure uses source-code groups rather than individual contracts as the splitting unit. Let

$$\mathcal{G} = \{g_1, g_2, \ldots, g_M\} \tag{34}$$

denote the set of unique SHA-256 groups. The group table contains one class label for each unique group. The implementation verifies that no group contains conflicting labels.

The maximum feasible number of folds is determined as:

$$K = \min\left(K_{\text{requested}}, N_{\text{proxy groups}}, N_{\text{nonproxy groups}}\right). \quad (35)$$

In this study, $K_{\text{requested}} = 3$. Stratified K-fold splitting is applied at the unique-group level. The selected groups are subsequently expanded into their corresponding contract indices.

For each outer fold, the dataset is divided into outer-training and testing partitions. An additional grouped split is then performed within the outer-training partition to create training and validation subsets. Leakage safety requires

$$\mathcal{G}_{\text{train}} \cap \mathcal{G}_{\text{validation}} = \emptyset, \quad (36)$$

$$\mathcal{G}_{\text{train}} \cap \mathcal{G}_{\text{test}} = \emptyset, \quad (37)$$

and:

$$\mathcal{G}_{\text{validation}} \cap \mathcal{G}_{\text{test}} = \emptyset. \quad (38)$$

Thus, identical contracts cannot appear in more than one partition.

Class-Weighted Model Training

Because the dataset is imbalanced, class weights are calculated using only the training subset. For class c, the weight is

$$w_c = \frac{N_{\text{train}}}{2\max(N_c, 1)}, \quad (39)$$

$N_{\text{train}}$is the number of training samples and $N_c$is the number of samples belonging to class c.

The weighted cross-entropy loss is

$$\mathcal{L} = -\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} w_{y_i} \log P(y_i \mid G_i). \quad (40)$$

The models are optimized using AdamW with a learning rate of $10^{-3}$and weight decay of 10^{-4}. The batch size is 16, the hidden dimension is 64, and the dropout probability is 0.30. Each model is trained for a maximum of 100 epochs.

Early stopping is based primarily on validation PR-AUC rather than validation loss. Let AP_tdenote the validation PR-AUC obtained at epoch t. The model state is updated when

$$AP_t > AP_{\text{best}} + 10^{-6}. \quad (41)$$

If validation PR-AUC cannot be calculated, balanced accuracy is used as the fallback selection score. Training is terminated when no improvement occurs for 15 consecutive epochs.

Performance Evaluation

Model performance is measured using Accuracy, Balanced Accuracy, Precision, Recall, F1-score, Matthews Correlation Coefficient, ROC-AUC, and

PR-AUC.

Accuracy is defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{42}$$

Balanced Accuracy is

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right). \tag{43}$$

Precision is

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{44}$$

while Recall is

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{45}$$

The F1-score is the harmonic mean of Precision and Recall:

$$F1 = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{46}$$

The Matthews Correlation Coefficient is

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{47}$$

ROC-AUC measures the model's ability to rank positive instances above negative instances across different classification thresholds. PR-AUC is calculated as the average precision over the Precision-Recall curve:

$$\text{PR-AUC} = \sum_{n} (R_n - R_{n-1})\, P_n, \tag{48}$$

$P_n$and $R_n$denote Precision and Recall at the n-th threshold. PR-AUC is particularly relevant in this study because proxy contracts form the minority class.

Results

Dataset Characteristics

The final dataset consisted of 377 Solidity smart contracts, including 360 non-proxy contracts and 17 proxy contracts, resulting in a proxy prevalence of 4.5%. The substantial difference in the number of samples between the two classes indicates a highly imbalanced classification problem, where proxy contracts represent only a small fraction of the available data. Such imbalance presents additional challenges for machine learning models because they tend to favor the majority class during training. Furthermore, the limited number of proxy

contracts emphasizes the importance of selecting appropriate evaluation metrics and validation strategies to obtain reliable performance estimates.

After grouping identical Solidity source codes using SHA-256 hashing, the dataset was reduced to 146 unique source-code groups, consisting of 139 non-proxy groups and 7 proxy groups. This reduction reveals that approximately 61.3% of the original contracts were duplicated source-code instances. The presence of a large number of duplicated contracts suggests that conventional random train-test splitting could place identical or nearly identical contracts into different data partitions, leading to information leakage and overly optimistic evaluation results. Therefore, leakage-safe grouped cross-validation was adopted to ensure that contracts derived from the same source-code group were assigned exclusively to a single partition during model evaluation.

Figure 2 summarizes the main characteristics of the dataset. Figure 2 presents the distribution of contracts at the individual contract level, while figure 2 illustrates the distribution after grouping duplicated source codes into unique groups. The comparison between these two distributions highlights the significant reduction in dataset size after duplicate removal and further justifies the use of group-aware evaluation. Figure 2 shows the distribution of Solidity source-code lengths for both classes. The results indicate that proxy contracts generally contain shorter source code than non-proxy contracts, reflecting their relatively simple implementation, which primarily consists of delegation and upgrade logic rather than complete business functionality. This difference in source-code length also suggests that the proposed graph representation captures structural characteristics from contracts with varying levels of complexity.

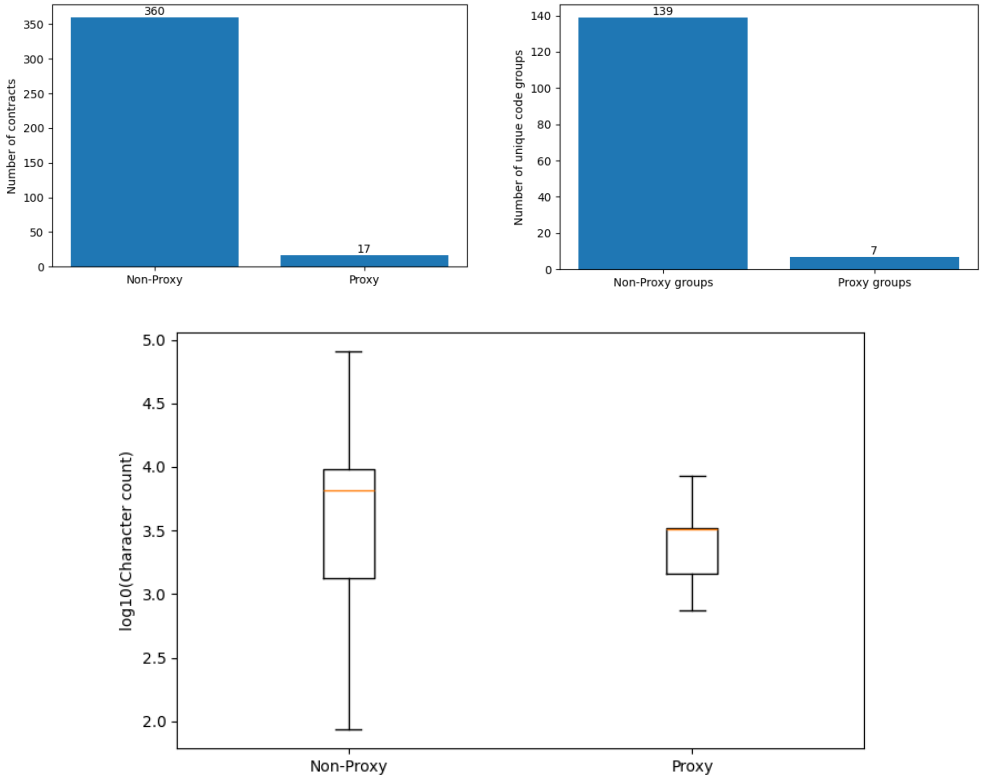


Figure 2 Dataset characteristics: contract-level class distribution, group-level class

distribution after duplicate removal, and Solidity source-code length distribution.

Performance Comparison

To ensure a fair and reliable evaluation, all experiments were conducted using leakage-safe grouped cross-validation, where contracts sharing the same SHA-256 source-code hash were assigned to the same data partition. This evaluation strategy was adopted to prevent duplicated Solidity contracts from appearing simultaneously in the training, validation, and testing sets, which could otherwise lead to overly optimistic performance estimates. The leakage audit confirmed that there was no overlap of source-code groups across all partitions in every cross-validation fold, indicating that the trained models were evaluated exclusively on previously unseen source-code groups. Consequently, the reported results reflect the models' ability to generalize to new smart contracts rather than memorizing duplicated code patterns.

The average classification performance obtained from the leakage-safe grouped cross-validation is summarized in table 1. Among the evaluated Graph Neural Network models, GCN achieved the highest ROC-AUC (0.851) and PR-AUC (0.606), demonstrating the strongest capability for ranking proxy and non-proxy contracts across different decision thresholds. In comparison, GraphSAGE produced the highest precision (0.365), recall (0.500), F1-score (0.279), and Matthews Correlation Coefficient (0.312), indicating a better balance between correctly identifying proxy contracts and maintaining overall classification performance. Although GAT achieved a competitive ROC-AUC of 0.815, its lower precision, F1-score, and MCC suggest that it was less effective in distinguishing the minority proxy class. Overall, these results indicate that GCN provides superior ranking performance, whereas GraphSAGE offers a more balanced classification performance under the leakage-safe evaluation framework.

Table 1 Performance comparison of Graph Neural Network models under leakage-safe grouped cross-validation.

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1-score	MCC	ROC-AUC	PR-AUC
GCN	0.947	0.662	0.200	0.333	0.250	0.255	0.851	0.606
GAT	0.770	0.553	0.047	0.333	0.082	0.070	0.815	0.551
GraphSAGE	0.829	0.681	0.365	0.500	0.279	0.312	0.729	0.330

Figure 3 provides a visual comparison of the classification performance achieved by the three evaluated Graph Neural Network models across all evaluation metrics. The figure complements the quantitative results presented in table 1 by illustrating the relative strengths and weaknesses of each model. GCN consistently achieved the highest values for ROC-AUC and PR-AUC, indicating superior capability in ranking proxy and non-proxy smart contracts under severe class imbalance. In contrast, GraphSAGE outperformed the other models in terms of precision, recall, F1-score, and Matthews Correlation Coefficient, demonstrating a better balance between correctly identifying proxy contracts and minimizing classification errors. Meanwhile, GAT generally produced lower performance across most evaluation metrics, suggesting that its attention mechanism was less effective for the relatively small and highly imbalanced dataset used in this study. Overall, the visual comparison confirms that each Graph Neural Network architecture exhibits different performance

characteristics, with GCN excelling in ranking-based evaluation and GraphSAGE providing more balanced classification performance.

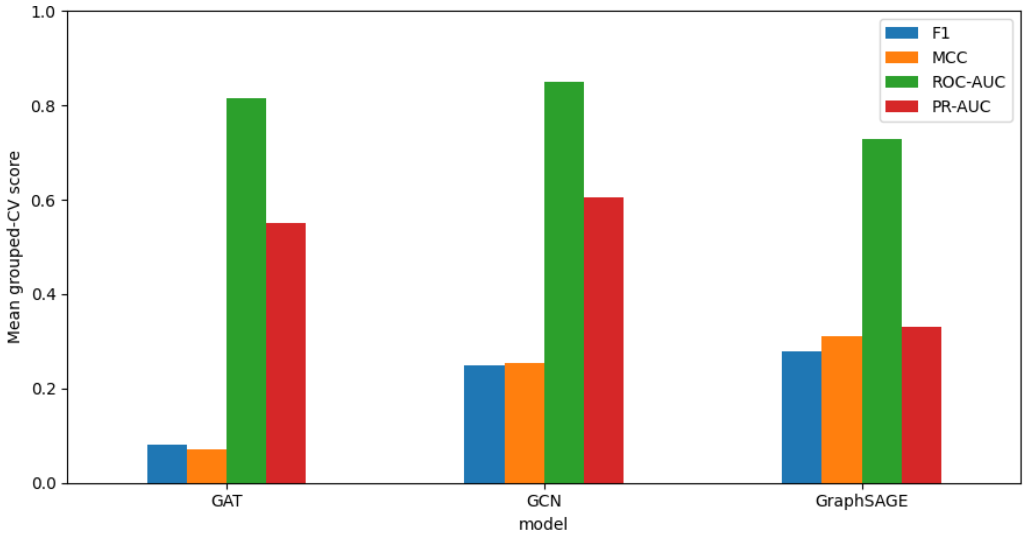


Figure 3 Performance comparison of GCN, GAT, and GraphSAGE under leakage-safe grouped cross-validation.

Classification Performance

To further assess the discriminative capability of the proposed graph-based proxy smart contract detection approach, Receiver Operating Characteristic (ROC) and Precision–Recall (PR) analyses were conducted using out-of-fold predictions obtained from leakage-safe grouped cross-validation. These evaluation methods provide a comprehensive assessment of model performance across different classification thresholds rather than relying on a single threshold-dependent metric. Unlike accuracy or F1-score, ROC and PR analyses evaluate how well a model separates positive and negative classes over the entire range of prediction scores.

Considering that proxy contracts account for only 4.5% of the dataset, the classification task is characterized by a severe class imbalance. Under such conditions, ROC-AUC alone may not fully reflect the effectiveness of a model because a high true negative rate can produce optimistic results even when the minority class is poorly detected. Therefore, Precision–Recall analysis was also performed, as it focuses specifically on the trade-off between precision and recall for the positive class. The combination of ROC and PR analyses provides complementary perspectives for evaluating the capability of the proposed models in identifying proxy contracts.

Figure 4 compares the ROC curves of the evaluated Graph Neural Network models, including GCN, GAT, and GraphSAGE. The curves illustrate the relationship between the true positive rate and the false positive rate over different classification thresholds. Among the evaluated models, GCN achieved the largest area under the ROC curve, indicating the strongest overall discrimination capability between proxy and non-proxy smart contracts. Although GAT also achieved a relatively high ROC-AUC, its overall classification performance remained lower than GCN because its threshold-dependent metrics, such as precision and F1-score, were considerably lower. GraphSAGE produced a slightly lower ROC-AUC but demonstrated competitive

classification performance based on other evaluation metrics.

Figure 5 presents the corresponding Precision–Recall curves for all evaluated models. The PR curves demonstrate that GCN consistently maintained higher precision across a broad range of recall values, resulting in the highest PR-AUC among the evaluated architectures. GraphSAGE achieved higher recall in several operating regions, indicating a greater ability to identify proxy contracts, although this improvement was accompanied by reduced precision. Meanwhile, GAT exhibited larger fluctuations in precision as recall increased, reflecting less stable performance on the highly imbalanced dataset. Overall, the ROC and Precision–Recall analyses confirm that the proposed graph-based structural representation enables effective proxy smart contract detection, with GCN providing the strongest ranking performance under the leakage-safe evaluation framework.

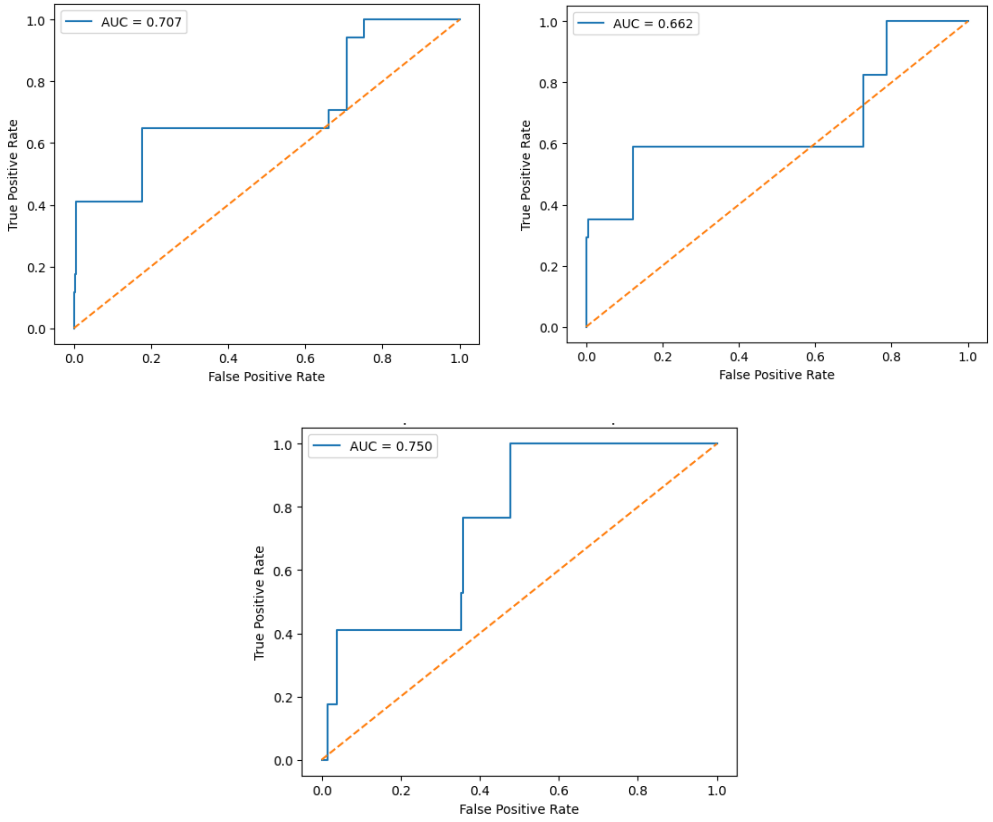


Figure 4 ROC curves of the evaluated Graph Neural Network models: GCN, GAT, and GraphSAGE.

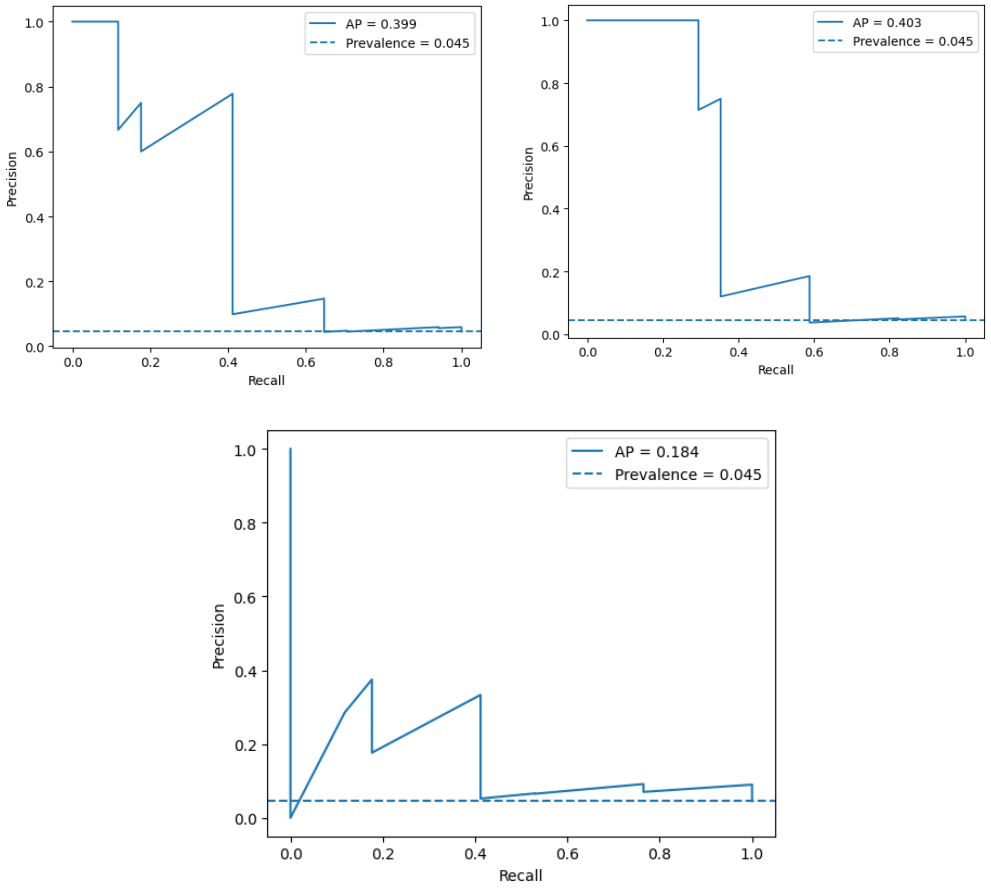


Figure 5 Precision–Recall curves of the evaluated Graph Neural Network models: GCN, GAT, and GraphSAGE.

The ROC analysis indicates that GCN achieved the strongest discrimination capability among the evaluated models. Similarly, the Precision–Recall analysis demonstrates that GCN consistently maintained higher precision across different recall levels, resulting in the highest average PR-AUC under leakage-safe grouped cross-validation. These findings suggest that the proposed structural graph representation enables Graph Neural Networks to effectively distinguish proxy contracts despite the highly imbalanced dataset.

Discussion

The experimental findings demonstrate that the proposed Solidity structural graph representation provides an effective foundation for proxy smart contract detection. Unlike conventional feature extraction methods that primarily rely on lexical tokens or sequential code representations, the proposed approach models the structural relationships among program elements, allowing Graph Neural Networks to learn dependencies embedded within the contract architecture. Proxy smart contracts are generally characterized by delegation mechanisms, inheritance structures, and interactions between functions instead of distinctive keywords or programming patterns. Consequently, preserving structural information enables the learning models to recognize architectural characteristics that differentiate proxy contracts from conventional smart contracts. The superior ROC-AUC and PR-AUC achieved by GCN suggest that graph convolution is particularly effective in capturing these structural patterns,

enabling the model to rank proxy and non-proxy contracts more accurately across different decision thresholds. Although GraphSAGE achieved higher precision, recall, F1-score, and Matthews Correlation Coefficient, its lower ROC-AUC and PR-AUC indicate that its overall ranking capability was less consistent than that of GCN. These observations suggest that different Graph Neural Network architectures emphasize different aspects of graph representation learning, resulting in distinct trade-offs between ranking performance and classification balance [21].

Another important contribution of this study is the implementation of leakage-safe grouped cross-validation, which directly addresses the high degree of source-code duplication within the dataset. The grouping process reduced the original dataset from 377 contracts to only 146 unique source-code groups, indicating that more than sixty percent of the contracts shared identical implementations. In smart contract repositories, code duplication commonly occurs because developers frequently deploy contracts derived from standard templates, open-source libraries, or previously audited implementations. If duplicated contracts are randomly distributed across the training and testing sets, machine learning models may simply memorize repeated code structures instead of learning generalizable features, leading to inflated performance estimates. The leakage audit confirmed that no source-code groups overlapped between the training, validation, and testing partitions throughout all cross-validation folds, ensuring that every evaluation was performed on previously unseen contract groups. Therefore, the reported performance more accurately reflects the ability of the proposed models to generalize to new smart contract implementations. This evaluation strategy improves the reliability and reproducibility of the experimental results and represents an important methodological contribution for future research on machine learning-based blockchain security analysis [22].

The proposed framework also has practical implications for blockchain security within Metaverse applications. Modern Metaverse platforms rely extensively on blockchain technology to manage digital assets, virtual land ownership, decentralized marketplaces, governance mechanisms, and other smart contract-based services. Many of these systems employ upgradeable proxy contracts to facilitate software maintenance and feature evolution without replacing deployed contract addresses. While this architecture improves flexibility, it also introduces additional complexity that can complicate security auditing and vulnerability assessment. The proposed graph-based proxy detection approach can serve as an automated preprocessing stage that identifies upgradeable contracts before detailed security analysis is performed. By automatically distinguishing proxy contracts from conventional contracts, security analysts can apply more specialized inspection techniques that account for delegation logic and implementation contracts, thereby improving the efficiency and accuracy of blockchain security assessments [23]. Nevertheless, this study has several limitations. The dataset contains only 17 proxy contracts, resulting in a highly imbalanced learning problem that may affect model generalization. In addition, the proposed representation is constructed solely from Solidity source code and does not incorporate runtime behavior, bytecode semantics, or transaction-level information. Future research may overcome these limitations by constructing larger and more diverse datasets, integrating multiple program representations such as Abstract Syntax Trees, Control Flow Graphs, and Call Graphs, and exploring more advanced graph learning

architectures, including Graph Transformers and heterogeneous Graph Neural Networks, to further improve proxy smart contract detection in increasingly complex blockchain and Metaverse ecosystems [24], [25].

Conclusion

This study proposed a Graph Neural Network-based framework for proxy smart contract detection using a Solidity structural graph representation and a leakage-safe evaluation strategy. Solidity source code was transformed into graph structures that preserve syntactic and structural relationships among program elements, enabling Graph Neural Network models to learn architectural characteristics associated with proxy contracts. To ensure reliable performance evaluation, leakage-safe grouped cross-validation was employed to prevent duplicated source-code groups from appearing across training, validation, and testing partitions. Experimental results demonstrated that the proposed framework effectively distinguished proxy and non-proxy smart contracts despite the highly imbalanced dataset. Among the evaluated models, GCN achieved the highest ROC-AUC and PR-AUC, indicating superior ranking capability, whereas GraphSAGE obtained the highest precision, recall, F1-score, and Matthews Correlation Coefficient, demonstrating a better balance between correctly identifying proxy contracts and minimizing classification errors. These findings confirm that graph-based structural representations provide meaningful information for proxy smart contract detection and that leakage-safe evaluation produces more reliable estimates of model generalization.

The proposed framework contributes to blockchain security research by combining structural graph learning with a rigorous evaluation methodology that mitigates information leakage caused by duplicated smart contracts. This contribution is particularly relevant for Metaverse applications, where upgradeable proxy contracts are widely adopted to support the continuous evolution of decentralized services while maintaining contract addresses. By automatically identifying proxy contracts before vulnerability assessment, the proposed approach can improve the efficiency of blockchain security auditing and support the development of more secure decentralized applications. Nevertheless, this study is limited by the relatively small number of proxy contracts and the use of structural information derived solely from Solidity source code. Future research should investigate larger and more diverse datasets, incorporate additional program representations such as Abstract Syntax Trees, Control Flow Graphs, and Call Graphs, and explore more advanced graph learning architectures to further improve proxy smart contract detection and enhance blockchain security in large-scale Metaverse ecosystems.

Declarations

Author Contributions

Conceptualization: S.A.N., K.M.; Methodology: K.M.; Software: S.A.N.; Validation: S.A.N., K.M.; Formal Analysis: S.A.N., K.M.; Investigation: S.A.N.; Resources: K.M.; Data Curation: K.M.; Writing Original Draft Preparation: S.A.N., K.M.; Writing Review and Editing: K.M., S.A.N.; Visualization: S.A.N.; All authors have 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] Y. Wang, Z. Su, N. Zhang, R. Xing, D. Liu, T. H. Luan, and X. Shen, "A survey on metaverse: Fundamentals, security, and privacy," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 319–352, 2023, doi: 10.1109/COMST.2022.3202047.

[2] A. M. Ebrahimi, B. Adams, G. A. Oliva, and A. E. Hassan, "UPC Sentinel: An accurate approach for detecting upgradeability proxy contracts in Ethereum," *Empirical Software Engineering*, vol. 30, no. 3, Art. no. 61, 2025, doi: 10.1007/s10664-024-10609-7.

[3] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 2020, no. July, pp. 3283–3290, 2020, doi: 10.24963/ijcai.2020/454.

[4] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, vol. 2019, no. October, pp. 143–153, 2019, doi: 10.1145/3359591.3359735.

[5] V. T. Truong, L. B. Le, and D. Niyato, "Blockchain meets metaverse and digital asset management: A comprehensive survey," *IEEE Access*, vol. 11, no. March, pp. 26258–26288, 2023, doi: 10.1109/ACCESS.2023.3257029.

[6] Z. Wei, J. Sun, Z. Zhang, X. Zhang, X. Yang, and L. Zhu, "Survey on quality assurance of smart contracts," *ACM Computing Surveys*, vol. 57, no. 2, Art. no. 32, pp. 1-36, 2025, doi: 10.1145/3695864.

[7] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Computing Surveys*, vol. 54, no. 7, Art. no. 148, pp. 1-38, 2022, doi: 10.1145/3464421.

[8] Y. Wang, J. He, N. Zhu, Y. Yi, Q. Zhang, H. Song, and R. Xue, "Security enhancement technologies for smart contracts in the blockchain: A survey," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 12, Art. no. e4341, 2021, doi: 10.1002/ett.4341.

[9] J. Cai, B. Li, J. Zhang, X. Sun, and B. Chen, “Combine sliced joint graph with graph neural networks for smart contract vulnerability detection,” *Journal of Systems and Software*, vol. 195, no. January, Art. no. 111550, 2023, doi: 10.1016/j.jss.2022.111550.

[10] J. Huang, S. Han, W. You, W. Shi, B. Liang, J. Wu, and Y. Wu, “Hunting vulnerable smart contracts via graph embedding based bytecode matching,” *IEEE Transactions on Information Forensics and Security*, vol. 16, no. January, pp. 2144–2156, 2021, doi: 10.1109/TIFS.2021.3050051.

[11] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan, and L. Jiang, “MANDO-GURU: Vulnerability detection for smart contract source code by heterogeneous graph embeddings,” in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, vol. 2022, no. November, pp. 1736–1740, 2022, doi: 10.1145/3540250.3558927.

[12] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, vol. 2017, no. February, pp. 1-14, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in *Proceedings of the International Conference on Learning Representations (ICLR)*, vol. 2018, no. January, pp. 1-12, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[14] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems 30*, vol. 2017, no. June, pp. 1024–1034, 2017, [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[15] H. H. Nguyen, N.-M. Nguyen, C. Xie, Z. Ahmadi, D. Kudendo, T.-N. Doan, and L. Jiang, “MANDO: Multi-level heterogeneous graph embeddings for fine-grained detection of smart contract vulnerabilities,” in *Proceedings of the 2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, 2022, vol. 2023, no. February, pp. 1–10, doi: 10.1109/DSAA54385.2022.10032337.

[16] S. Kapoor and A. Narayanan, “Leakage and the reproducibility crisis in machine-learning-based science,” *Patterns*, vol. 4, no. 9, Art. no. 100804, 2023, doi: 10.1016/j.patter.2023.100804.

[17] T. Huynh-The, T. R. Gadekallu, W. Wang, G. Yenduri, P. Ranaweera, Q.-V. Pham, D. B. da Costa, and M. Liyanage, “Blockchain for the metaverse: A review,” *Future Generation Computer Systems*, vol. 143, no. June, pp. 401–419, 2023, doi: 10.1016/j.future.2023.02.008.

[18] S. Casale-Brunet, M. Mattavelli, and L. Chiariglione, “Exploring blockchain-based metaverses: Data collection and valuation of virtual lands using machine learning techniques,” *Digital Business*, vol. 3, no. 2, Art. no. 100068, pp. 1-14, 2023, doi: 10.1016/j.digbus.2023.100068.

[19] Z. Zheng, S. Xie, H. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, “An overview on smart contracts: Challenges, advances and platforms,” *Future Generation Computer Systems*, vol. 105, no. April, pp. 475–491, 2020, doi: 10.1016/j.future.2019.12.019.

[20] C. Wu, J. Xiong, H. Xiong, Y. Zhao, and W. Yi, “A review on recent progress of

smart contract in blockchain," *IEEE Access*, vol. 10, no. May, pp. 50839–50863, 2022, doi: 10.1109/ACCESS.2022.3174052.

[21] J. Cai, B. Li, J. Zhang, X. Sun, and B. Chen, "Extended abstract of combine sliced joint graph with graph neural networks for smart contract vulnerability detection," in *Proceedings of the 30th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, vol. 2023, no. May, pp. 851–852, 2023, doi: 10.1109/SANER56733.2023.00101.

[22] N. Alturayeif and J. Hassine, "Data leakage detection in machine learning code: Transfer learning, active learning, or low-shot prompting?" *PeerJ Computer Science*, vol. 11, no. March, Art. no. e2730, 2025, doi: 10.7717/peerj-cs.2730.

[23] I. Qasse, M. Hamdaqa, and B. Þ. Jónsson, "Immutable in principle, upgradeable by design: Exploratory study of smart contract upgradeability," *arXiv preprint* arXiv:2407.01493, vol. 2024, no. July, pp. 1-45, 2024, doi: 10.48550/arXiv.2407.01493.

[24] B. Hu, Z. Zhang, J. Liu, Y. Liu, J. Yin, R. Lu, and X. Lin, "A comprehensive survey on smart contract construction and execution: Paradigms, tools, and systems," *Patterns*, vol. 2, no. 2, Art. no. 100179, 2021, doi: 10.1016/j.patter.2020.100179.

[25] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *Journal of Network and Computer Applications*, vol. 177, no. March, Art. no. 102857, 2021, doi: 10.1016/j.jnca.2020.102857.