Leakage-Safe Proxy Smart Contract Detection with CodeBERT Embeddings, Static Solidity Features, and Classical Machine Learning Models

R. Rajasree[1,*], K. Stalin[2]

[1,2]Department of EEE, AMET University, Chennai, India; ² Department of Marine, AMET University, Chennai, India

ABSTRACT

Proxy smart contracts are widely adopted in Ethereum to enable contract upgradeability while preserving deployed contract addresses. However, automatically identifying proxy contracts remains challenging due to the limited availability of labeled datasets, severe class imbalance, and the high prevalence of duplicated source code, which can introduce information leakage during model evaluation. This study proposes a leakage-safe proxy smart contract detection framework by combining pretrained CodeBERT embeddings, handcrafted static Solidity features, and classical machine learning classifiers under a Stratified Group Cross Validation evaluation protocol. The proposed grouping strategy assigns contracts with identical normalized source code to the same cross-validation fold, preventing duplicated contracts from appearing simultaneously in the training and testing sets. Experiments were conducted on a dataset containing 377 Ethereum smart contracts, comprising 360 non-proxy contracts and 17 proxy contracts, which were grouped into 146 unique source-code groups. Three classifiers, namely Logistic Regression, Random Forest, and XGBoost, were evaluated using Accuracy, Balanced Accuracy, Precision, Recall, F1-score, ROC-AUC, Precision–Recall Area Under the Curve (PR-AUC), and Matthews Correlation Coefficient (MCC). Experimental results show that Logistic Regression achieved the best overall performance, obtaining a Balanced Accuracy of 81.38%, Precision of 61.11%, Recall of 64.71%, F1-score of 62.86%, PR-AUC of 0.4977, and MCC of 0.6108. The leakage audit further confirmed zero source-code group overlap between the training and testing partitions across all cross-validation folds. These findings demonstrate that integrating pretrained code embeddings with leakage-safe group-aware evaluation provides a reliable and effective approach for proxy smart contract detection while producing more realistic estimates of model generalization in the presence of duplicated smart contract source code.

Keywords Proxy Smart Contract Detection, CodeBERT, Ethereum Smart Contracts, Group-Aware Cross Validation, Information Leakage

Submitted 3 June 2026
Accepted 18 August 2026
Published 17 November 2026

Corresponding author
R. Rajasree,
raji.sree1988@gmail.com

Additional Information and Declarations can be found on page 294

DOI: 10.47738/jcrb.v3i4.85

Copyright
2026 Rajasree and Stalin

Distributed under
Creative Commons CC-BY 4.0

INTRODUCTION

Blockchain technology has transformed the way decentralized applications are developed by enabling trustless execution of programmable transactions through smart contracts. Among various blockchain platforms, Ethereum has become the most widely adopted ecosystem for decentralized applications because it supports Turing-complete smart contracts written primarily in Solidity. Smart contracts automate contractual agreements without relying on intermediaries, making them widely used in decentralized finance, non-fungible tokens, decentralized autonomous organizations, and other blockchain-based services. As blockchain applications continue to evolve, the need to modify deployed smart contracts without changing their contract addresses has led to

How to cite this article: R. Rajasree and K. Stalin, "Leakage-Safe Proxy Smart Contract Detection with CodeBERT Embeddings, Static Solidity Features, and Classical Machine Learning Models," *J. Curr. Res. Blockchain*, vol. 3, no. 4, pp. 279-297, 2026.

the widespread adoption of proxy contracts. Proxy smart contracts separate contract logic from contract storage, allowing implementation contracts to be upgraded while preserving the original contract address. This upgradeability mechanism improves maintainability but also increases the complexity of smart contract analysis because proxy contracts often differ structurally from conventional contracts [1].

Automatically identifying proxy smart contracts has become increasingly important for blockchain analysis, security auditing, vulnerability assessment, and software maintenance. Accurate proxy contract detection enables developers and auditors to correctly interpret contract architectures before performing further analyses, such as vulnerability detection or behavioral inspection. However, proxy contract identification remains a challenging task because proxy contracts represent only a small proportion of deployed contracts and exhibit diverse implementation patterns. Furthermore, many deployed contracts are derived from previously published implementations, resulting in extensive source-code duplication throughout the Ethereum ecosystem. Such duplication increases the risk of information leakage during machine learning evaluation if identical contracts are unintentionally included in both the training and testing datasets [2].

Recent advances in machine learning have significantly improved automated source-code analysis. Traditional approaches primarily relied on handcrafted software metrics and manually engineered static features, whereas more recent studies have adopted pretrained code representation models capable of capturing semantic and syntactic information directly from source code. Among these models, CodeBERT has demonstrated strong performance across a variety of software engineering tasks because it produces contextual embeddings that preserve programming language semantics without requiring handcrafted feature engineering. In the context of smart contract analysis, pretrained code embeddings have shown promising capability for representing Solidity programs and improving classification performance. Nevertheless, most existing studies primarily focus on vulnerability detection, malicious contract identification, or code classification rather than proxy smart contract detection. Moreover, many existing approaches emphasize improvements in model architectures while paying relatively little attention to the reliability of the evaluation protocol itself [3].

Although previous studies have demonstrated encouraging classification performance, several limitations remain. First, existing research generally assumes that each contract represents an independent sample, despite the fact that smart contract repositories contain a substantial amount of duplicated source code due to contract cloning and template reuse. Conventional random train-test splitting or standard cross-validation may therefore introduce information leakage by allowing identical contracts to appear simultaneously in the training and testing partitions. Under these conditions, classification models may partially memorize duplicated implementations instead of learning characteristics that generalize to previously unseen contracts, resulting in overly optimistic performance estimates. Second, research specifically addressing proxy smart contract detection remains relatively limited compared with other smart contract analysis tasks. Third, relatively few studies investigate whether pretrained code embeddings can effectively detect proxy contracts using lightweight classical machine learning models under leakage-safe evaluation protocols [4].

Based on these observations, the primary research gap addressed in this study

is the absence of a leakage-safe evaluation framework for proxy smart contract detection that explicitly considers duplicated Solidity source code during model validation. While previous studies have mainly concentrated on developing increasingly sophisticated learning models, limited attention has been given to preventing duplicate-code leakage that may compromise evaluation reliability. Consequently, it remains unclear whether reported performance truly reflects a model's ability to generalize to previously unseen proxy contract implementations.

To address this gap, this study proposes a proxy smart contract detection framework that combines pretrained CodeBERT embeddings with handcrafted static Solidity features and classical machine learning classifiers under a Stratified Group Cross Validation evaluation protocol. Contracts sharing identical normalized source code are assigned exclusively to the same cross-validation fold, ensuring that duplicated implementations never appear simultaneously in the training and testing partitions. Three classifiers, namely Logistic Regression, Random Forest, and XGBoost, are evaluated using multiple performance metrics suitable for imbalanced classification problems, including Balanced Accuracy, Precision, Recall, F1-score, ROC-AUC, Precision-Recall Area Under the Curve (PR-AUC), and Matthews Correlation Coefficient (MCC). The main contribution of this study is the demonstration that leakage-safe group-aware evaluation can provide more reliable estimates of model generalization while maintaining effective proxy smart contract detection using pretrained semantic code representations and classical machine learning techniques [5].

Literature Review and Related Works

Blockchain technology has enabled the development of decentralized applications through smart contracts, which are self-executing programs deployed on blockchain platforms such as Ethereum. Because smart contracts are immutable after deployment, programming errors or design flaws may lead to permanent financial losses. Consequently, automated smart contract analysis has become an active research area aimed at improving software reliability, security, and maintainability [6].

Early studies primarily focused on static analysis, symbolic execution, and formal verification techniques to detect vulnerabilities in Solidity smart contracts. Although these approaches are effective for identifying predefined vulnerability patterns, they generally require manually designed rules and often struggle to generalize to unseen contract implementations [7].

To overcome these limitations, machine learning methods have been introduced to automatically learn discriminative patterns from smart contract source code. Traditional machine learning approaches commonly rely on handcrafted software metrics and static code features extracted from Solidity programs before applying classifiers such as Logistic Regression, Random Forest, or Gradient Boosting models [8].

Recent advances in deep learning have shifted attention toward pretrained code representation models. CodeBERT has emerged as one of the most widely adopted transformer-based models because it learns contextual representations of programming languages through large-scale pretraining. These semantic embeddings have demonstrated strong performance in various software engineering tasks, including code search, code summarization, defect

prediction, and vulnerability detection [9].

Several studies have further demonstrated that CodeBERT can effectively represent Solidity smart contracts for vulnerability detection. By combining semantic embeddings with traditional static features, machine learning classifiers are able to improve detection performance while reducing the need for extensive manual feature engineering [10].

Beyond semantic embeddings, structural information has also been incorporated into smart contract analysis. Control flow graphs, abstract syntax trees, and graph-based neural networks have been explored to capture relationships among program statements that cannot be represented using token sequences alone. These approaches generally improve representation quality but require substantially higher computational complexity during preprocessing and model training [11].

Machine learning algorithms remain an important component of automated smart contract analysis. While many recent studies emphasize deep neural networks, several investigations have shown that classical machine learning algorithms remain competitive when informative feature representations are available. Simpler classifiers frequently provide better interpretability, lower computational cost, and improved robustness when training data are relatively limited [12].

In addition to improving model architectures, several researchers have investigated smart contract classification problems beyond vulnerability detection. These studies include malicious contract identification, contract categorization, and functional classification, demonstrating that semantic source-code representations can successfully distinguish different contract behaviors [13].

Despite these advances, proxy smart contract detection has received relatively limited attention. Proxy contracts differ from conventional contracts because executable logic is separated from persistent storage, allowing contract implementations to be upgraded while preserving deployed addresses. This architectural characteristic introduces additional challenges for automated source-code analysis and classification [14].

Another characteristic of Ethereum smart contracts is the extensive reuse of source code. Large numbers of deployed contracts originate from publicly available templates and previously verified implementations, resulting in significant duplication within blockchain repositories [15].

The widespread existence of duplicated source code introduces an important challenge for machine learning evaluation. When identical contracts are randomly distributed between training and testing datasets, classifiers may memorize recurring implementations instead of learning generalized semantic patterns. Consequently, reported performance may become overly optimistic because of information leakage [16].

Several recent studies have emphasized the importance of constructing reliable benchmark datasets and adopting evaluation strategies that minimize overlap between training and testing samples. These studies suggest that improving evaluation methodology is equally important as improving model architecture because reliable evaluation directly affects the validity of reported performance [17].

Research has also demonstrated that pretrained code representation models can generalize effectively across different software engineering tasks without requiring task-specific architectural modifications. This finding indicates that pretrained embeddings provide transferable semantic information suitable for downstream classification problems [18].

Furthermore, clone detection studies have revealed that Solidity repositories contain a remarkably high proportion of duplicated contracts, reinforcing the necessity of considering source-code similarity during experimental evaluation [19].

Based on the existing literature, it can be observed that previous studies have primarily focused on improving feature representation and classification performance, whereas relatively limited attention has been given to preventing information leakage caused by duplicated source code during proxy smart contract detection. Therefore, this study proposes a leakage-safe proxy smart contract detection framework by combining pretrained CodeBERT embeddings, handcrafted static Solidity features, and Stratified Group Cross Validation to provide a more reliable evaluation of model generalization [20].

Methodology

This study proposes a leakage-safe framework for proxy smart contract detection by integrating pretrained CodeBERT embeddings, handcrafted static Solidity features, and classical machine learning classifiers under a Stratified Group Cross Validation evaluation protocol. Unlike conventional machine learning pipelines that randomly partition contracts into training and testing datasets, the proposed framework prevents information leakage by ensuring that contracts sharing identical normalized source code are assigned exclusively to the same validation fold. As illustrated in figure 1, the proposed framework consists of six sequential stages: (1) dataset preparation, (2) source code normalization and duplicate grouping, (3) feature extraction using CodeBERT and handcrafted Solidity features, (4) feature fusion, (5) leakage-safe model training using Stratified Group Cross Validation with inner threshold optimization, and (6) performance evaluation. This workflow ensures that duplicate contracts are never simultaneously assigned to the training and testing partitions, thereby providing a more reliable estimate of model generalization.

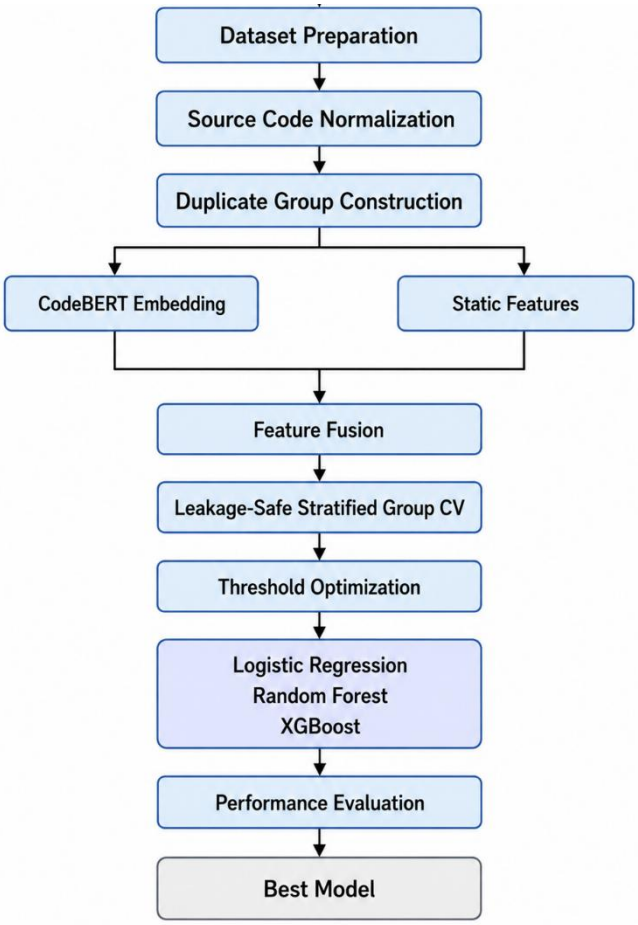

Figure 1 Research Framework

Dataset Preparation

The experimental dataset consists of Ethereum smart contracts represented by Solidity source code together with binary class labels, where proxy contracts are assigned label 1 and non-proxy contracts are assigned label 0. Invalid labels, missing source code, and incomplete records are removed during preprocessing. The cleaned Solidity source code is then selected as the primary textual input for feature extraction.

Source Code Normalization and Duplicate Group Construction

To prevent information leakage, each Solidity contract is first normalized by removing comments and whitespace before constructing duplicate groups. The normalized source code is represented as:

$$C_i^{norm} = Normalize(C_i) \quad (1)$$

$C_i$denotes the original Solidity source code.

A SHA-256 hash is then generated for every normalized contract

$$G_i = SHA256(C_i^{norm}) \quad (2)$$

$G_i$represents the unique group identifier.

Contracts having identical group hashes are assigned to the same validation group,

$$G_i = G_j \tag{3}$$

indicating that contracts $i$and $j$originate from the same normalized source code. During Stratified Group Cross Validation, every group is allocated entirely to a single fold, preventing duplicated contracts from appearing simultaneously in the training and testing datasets.

Feature Extraction

The normalized Solidity source code is encoded using the pretrained CodeBERT model. Given the contextual token embeddings

$$H = \{h_1, h_2, \dots, h_n\}, \tag{4}$$

a fixed-length representation is obtained through mean pooling,

$$E = \frac{1}{n}\sum_{i=1}^{n} h_i \tag{5}$$

$h_i$denotes the contextual embedding of token i,

$n$represents the number of valid tokens,

$E$is the semantic embedding of the smart contract.

Mean pooling allows the semantic information from all valid tokens to contribute equally to the final representation instead of relying solely on the special classification token.

Besides semantic embeddings, handcrafted static features are extracted directly from the Solidity source code. These features describe structural characteristics such as function declarations, modifiers, delegatecall usage, fallback functions, storage variables, mappings, constructors, payable functions, and proxy-related keywords.

The resulting feature vector is expressed as:

$$S = [s_1, s_2, \dots, s_{26}] \tag{6}$$

each $s_i$corresponds to one handcrafted Solidity feature.

Feature Fusion

The semantic embedding produced by CodeBERT is concatenated with the handcrafted Solidity features to form the final input representation,

$$X = [E \;||\; S] \tag{7}$$

Edenotes the CodeBERT embedding,

Srepresents the handcrafted feature vector,

||denotes vector concatenation.

This hybrid representation combines semantic program understanding with explicit structural characteristics of Solidity contracts.

Classification Models

Three supervised machine learning algorithms are evaluated.

For Logistic Regression, the posterior probability of a proxy contract is estimated using the sigmoid function,

$$P(y = 1 \mid X) = \frac{1}{1 + e^{-(w^T X + b)}} \tag{8}$$

$w$denotes the model parameters and $b$is the bias.

Random Forest predicts the final class through majority voting among decision trees,

$$\hat{y} = \text{mode}(T_1, T_2, \ldots, T_K) \tag{9}$$

$T_k$denotes the prediction of the k-th decision tree.

For XGBoost, the prediction is expressed as:

$$\hat{y} = \sum_{k=1}^{K} f_k\,(X), \tag{10}$$

$f_k$denotes the regression tree learned during boosting iteration k. Class imbalance is handled by automatically computing the positive class weight based on the training data.

Leakage-Safe Stratified Group Cross Validation

Model evaluation employs Stratified Group Cross Validation. Let

$$\mathcal{G} = \{G_1, G_2, \ldots, G_m\} \tag{11}$$

be the set of unique source-code groups.

The partition satisfies

$$G_i \cap G_j = \emptyset \tag{12}$$

for every training and testing fold, ensuring that no duplicated source-code group appears in both datasets.

Within each outer fold, an inner grouped cross validation determines the optimal classification threshold,

$$t^* = \arg\max_t F1(t) \tag{13}$$

candidate thresholds range from 0.01 to 0.99.

The selected threshold is subsequently applied only to the corresponding outer testing fold, thereby preventing test information from influencing threshold selection.

Performance Evaluation

Model performance is evaluated using several metrics appropriate for imbalanced binary classification.

Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

Precision

$$Precision = \frac{TP}{TP + FP} \tag{15}$$

Recall

$$Recall = \frac{TP}{TP + FN} \tag{16}$$

F1-Score

$$F1 = \frac{2PR}{P + R} \tag{17}$$

Balanced Accuracy

$$BA = \frac{Sensitivity + Specificity}{2} \tag{18}$$

Matthews Correlation Coefficient (MCC)

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{19}$$

TP, TN, FP, and FNdenote true positives, true negatives, false positives, and false negatives, respectively.

In addition to these metrics, Receiver Operating Characteristic Area Under the Curve (ROC-AUC) and Precision–Recall Area Under the Curve (PR-AUC) are computed using prediction probabilities to assess ranking performance under class imbalance. Finally, the best-performing model is retrained using the complete dataset and exported together with the optimized decision threshold for future inference.

Results

Dataset Characteristics

The experimental dataset consists of 377 Ethereum smart contracts, including 360 non-proxy contracts and 17 proxy contracts, resulting in a highly imbalanced binary classification problem in which proxy contracts account for only 4.5% of all samples. Such class imbalance presents a significant challenge for supervised learning because classifiers tend to favor the majority class during training, potentially reducing their ability to correctly identify proxy contracts. In

addition to the imbalance, the dataset contains a substantial amount of duplicated source code due to contract redeployment and code reuse, which are common practices in the Ethereum ecosystem. Previous empirical studies have shown that smart contract development frequently involves code cloning and template reuse, making duplicated contracts a common characteristic of blockchain datasets Consequently, evaluating machine learning models using conventional cross-validation may lead to overly optimistic performance estimates if duplicated contracts are distributed across both the training and testing sets.

To address this issue, contracts were grouped using normalized SHA-256 hashes generated from their Solidity source code after removing formatting differences. This grouping process produced 146 unique source-code groups, consisting of 139 non-proxy groups and 7 proxy groups. The grouping results indicate that 267 out of 377 contracts (70.8%) share identical source code with at least one other contract, confirming the high degree of code duplication within the dataset. Figure 2 presents the class distribution at both the contract level and the grouped source-code level. Although grouping substantially reduced the number of independent samples from 377 contracts to 146 unique source-code groups, the severe class imbalance remained largely unchanged. These observations demonstrate that the proposed classification task is challenging not only because of the scarcity of proxy contracts but also because a reliable evaluation requires preventing information leakage caused by duplicated source code. Therefore, the use of group-aware cross-validation is essential to obtain a realistic estimate of model generalization performance on previously unseen smart contract implementations.

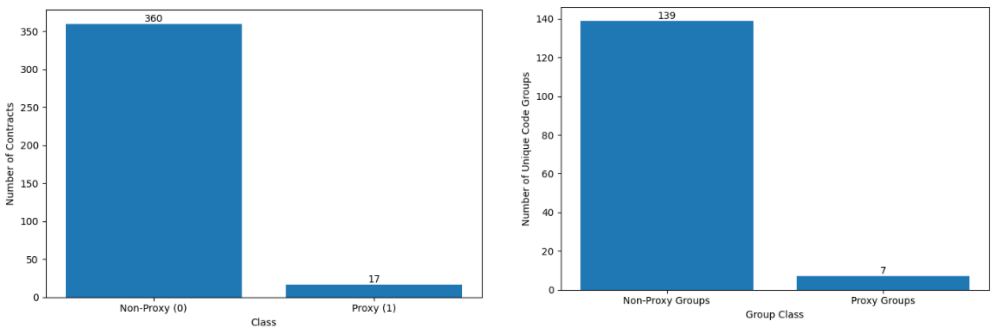

Figure 2 Class distribution of the experimental dataset: contract-level distribution and unique source-code group distribution.

Leakage Verification

To obtain a realistic estimate of model performance, this study employed Stratified Group Cross Validation as the primary evaluation strategy. Unlike conventional stratified cross-validation, which only preserves the class distribution across folds, Stratified Group Cross Validation additionally ensures that all contracts originating from the same normalized source code are assigned to a single fold. Each contract was grouped using a normalized SHA-256 hash of its Solidity source code, allowing contracts with identical implementations to be treated as one independent group. During cross-validation, the grouping constraint was enforced while maintaining a balanced distribution of proxy and non-proxy groups across folds as much as possible. This evaluation strategy was adopted because the dataset contains a large

proportion of duplicated source code, and allowing identical contracts to appear in both the training and testing sets would artificially inflate the reported classification performance.

The effectiveness of the grouping strategy was verified through a leakage audit performed after the cross-validation split. As shown in table 1, the number of overlapping source-code groups between the training and testing partitions was zero for every fold, confirming that each unique source-code group was evaluated only once during testing. This result demonstrates that the proposed evaluation protocol successfully eliminated information leakage caused by duplicated contracts and ensured that model performance was measured on previously unseen smart contract implementations rather than memorized source code. Although the number of training and testing groups varies across folds due to the limited availability of only seven proxy source-code groups, the absence of group overlap provides strong evidence that the reported performance metrics reflect the model's true generalization capability under realistic deployment conditions.

Table 1 Leakage audit of Stratified Group Cross Validation

Fold	Training Groups	Testing Groups	Group Overlap
1	143	3	0
2	110	36	0
3	111	35	0
4	110	36	0
5	110	36	0

Overall Classification Performance

Three classical machine learning classifiers, namely Logistic Regression, Random Forest, and XGBoost, were evaluated using the combined feature representation consisting of pretrained CodeBERT embeddings and handcrafted static Solidity features. All models were assessed using the proposed leakage-safe evaluation protocol based on Stratified Group Cross Validation to ensure that contracts sharing identical source code were not simultaneously included in the training and testing partitions. Performance was measured using multiple evaluation metrics, including Accuracy, Balanced Accuracy, Precision, Recall, F1-score, ROC-AUC, PR-AUC, and Matthews Correlation Coefficient (MCC). Since the dataset is highly imbalanced with only 17 proxy contracts, metrics such as Balanced Accuracy, PR-AUC, and MCC were considered more informative than overall Accuracy because they provide a better assessment of classifier performance on the minority class. The grouped out-of-fold evaluation results for all classifiers are presented in table 2.

Among the evaluated models, Logistic Regression achieved the best overall performance across the majority of evaluation metrics. The model obtained a Balanced Accuracy of 81.38%, Precision of 61.11%, Recall of 64.71%, F1-score of 62.86%, PR-AUC of 0.4977, and MCC of 0.6108, demonstrating a favorable balance between correctly identifying proxy contracts and minimizing false-positive predictions. Although XGBoost achieved the highest ROC-AUC of 0.8402, its substantially lower Precision, F1-score, PR-AUC, and MCC indicate that the model produced more false-positive predictions and exhibited weaker overall performance on the minority class. Similarly, Random Forest achieved the lowest performance across nearly all evaluation metrics, suggesting that it

was less effective in distinguishing proxy and non-proxy contracts under the severe class imbalance of the dataset. Based on these results, Logistic Regression was selected as the final classifier because it consistently provided the most balanced and reliable performance under the proposed group-aware evaluation protocol.

Table 2 Overall grouped out-of-fold classification performance

Model	Accuracy	Balanced Accuracy	Precision	Recall	F1-score	ROC-AUC	PR-AUC	MCC
Logistic Regression	0.9655	0.8138	0.6111	0.6471	0.6286	0.7644	0.4977	0.6108
XGBoost	0.8833	0.7708	0.2245	0.6471	0.3333	0.8402	0.3750	0.3342
Random Forest	0.8090	0.7038	0.1333	0.5882	0.2174	0.7732	0.1321	0.2119

Confusion Matrix Analysis

To further examine the classification behavior of the best-performing model, figure 3 presents the grouped out-of-fold confusion matrix obtained by the Logistic Regression classifier. The confusion matrix provides a detailed breakdown of prediction outcomes by showing the number of correctly and incorrectly classified samples for each class. Unlike aggregate evaluation metrics, the confusion matrix offers a more comprehensive understanding of how the classifier distinguishes between proxy and non-proxy contracts, particularly in the presence of severe class imbalance. Since proxy contracts account for only a small proportion of the dataset, analyzing the distribution of true positives, false positives, true negatives, and false negatives is essential for assessing the practical effectiveness of the proposed detection approach.

As illustrated in figure 3, the classifier correctly identified 353 non-proxy contracts and 11 proxy contracts, while incorrectly classifying 7 non-proxy contracts as proxy and failing to detect 6 proxy contracts. These prediction outcomes correspond to a Recall of 64.71%, indicating that approximately two-thirds of all proxy contracts were successfully detected. At the same time, the model maintained a relatively low number of false-positive predictions, with only seven non-proxy contracts being incorrectly labeled as proxy. This balance between correctly detecting proxy contracts and minimizing false alarms is further reflected in the achieved Precision of 61.11%, F1-score of 62.86%, and Matthews Correlation Coefficient (MCC) of 0.6108. Overall, the confusion matrix demonstrates that the proposed Logistic Regression model provides reliable classification performance under a highly imbalanced dataset while maintaining good discrimination between the two contract categories.

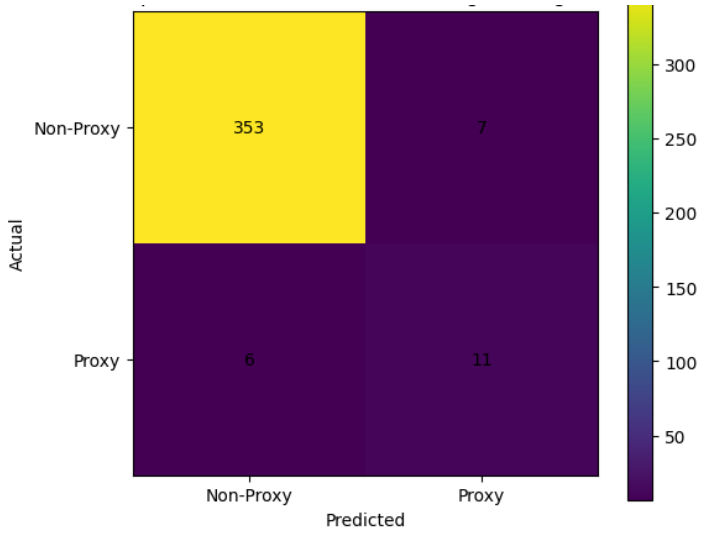

Figure 3 **Grouped out-of-fold confusion matrix of the Logistic Regression classifier.**

Precision–Recall Analysis

Because proxy contracts represent only 4.5% of the experimental dataset, evaluating the classifier using the Precision–Recall (PR) curve provides a more appropriate assessment than relying solely on the Receiver Operating Characteristic (ROC) curve. In highly imbalanced classification problems, ROC-AUC may present an overly optimistic view of model performance because the large number of true negatives dominates the evaluation. In contrast, the PR curve focuses on the relationship between Precision and Recall, making it more sensitive to the classifier's ability to correctly identify the minority class. Therefore, the PR curve offers a more meaningful evaluation of proxy contract detection, where accurately identifying the limited number of positive samples is considerably more important than correctly classifying the already dominant non-proxy class. Figure 4 illustrates the grouped out-of-fold Precision–Recall curve obtained from the best-performing Logistic Regression model.

As shown in figure 4, the proposed model achieved an Average Precision (AP) of 0.4977, which is substantially higher than the positive class prevalence of 0.045 in the dataset. This result indicates that the model is capable of ranking proxy contracts significantly better than random guessing, despite the extremely limited number of positive samples available for training and evaluation. Furthermore, the relatively high PR-AUC, together with a Precision of 61.11%, Recall of 64.71%, F1-score of 62.86%, and Matthews Correlation Coefficient (MCC) of 0.6108, demonstrates that the classifier achieves a balanced trade-off between detecting proxy contracts and limiting false-positive predictions. These findings indicate that combining pretrained CodeBERT embeddings with handcrafted static Solidity features enables the Logistic Regression classifier to learn discriminative representations that remain effective even under severe class imbalance and leakage-safe group-aware evaluation.

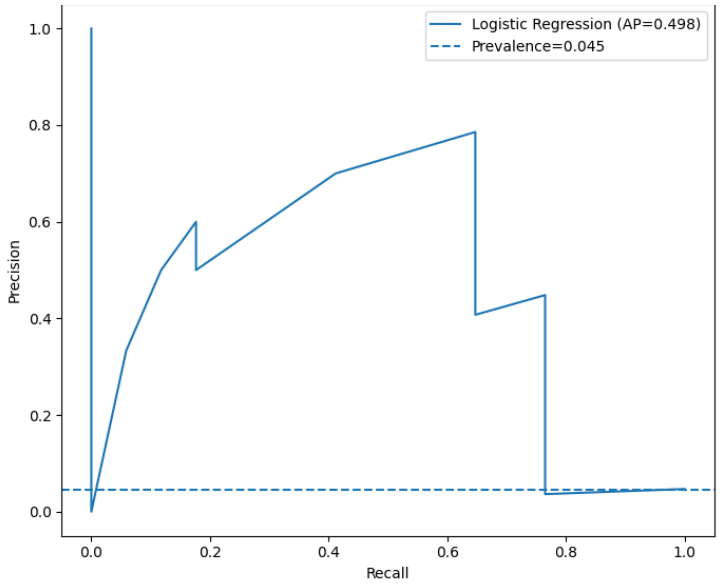

Figure 4 Grouped out-of-fold Precision–Recall curve of the Logistic Regression classifier.

Discussion

The experimental results demonstrate that the proposed approach is capable of effectively detecting proxy smart contracts by combining pretrained CodeBERT embeddings with handcrafted static Solidity features under a leakage-safe evaluation protocol. Among the evaluated classifiers, Logistic Regression consistently achieved the best overall performance across most evaluation metrics, including Balanced Accuracy, Precision, F1-score, PR-AUC, and Matthews Correlation Coefficient. Although XGBoost obtained a higher ROC-AUC, its lower Precision, F1-score, PR-AUC, and MCC indicate that it generated more false-positive predictions. These findings suggest that, for relatively small and highly imbalanced datasets, a simpler linear classifier can provide better generalization than more complex ensemble models. The semantic representations generated by the pretrained CodeBERT model appear to provide a sufficiently discriminative feature space, allowing Logistic Regression to effectively separate proxy and non-proxy contracts without requiring a more sophisticated classification model. In contrast, tree-based ensemble methods may be more susceptible to overfitting when the number of positive training samples is very limited, which is consistent with previous observations on pretrained code representations and lightweight classifiers for software engineering tasks [21].

Another important finding of this study is the effectiveness of the proposed leakage-safe evaluation protocol. More than seventy percent of the contracts in the dataset share identical source code with at least one other contract, indicating a substantial level of code duplication. This characteristic reflects the common practice of code reuse in Ethereum smart contract development, where developers frequently deploy contracts derived from existing implementations. If duplicated contracts are randomly distributed between the training and testing sets, the classifier may achieve artificially high performance by recognizing previously seen source code instead of learning patterns that generalize to unseen contracts. The leakage audit confirmed that no source-

code group appeared simultaneously in both partitions throughout the cross-validation process. Consequently, the reported evaluation results provide a more realistic estimate of the classifier's ability to detect proxy smart contracts in practical deployment scenarios where previously unseen contract implementations are encountered. This finding supports recent studies emphasizing that overlap-aware evaluation protocols are essential for obtaining reliable estimates of model performance when software repositories contain substantial code duplication [22].

The evaluation also highlights the importance of selecting appropriate performance metrics for highly imbalanced datasets. Since proxy contracts account for only a small proportion of the experimental data, overall accuracy alone is insufficient for assessing classifier effectiveness because it is strongly influenced by the majority class. For this reason, greater emphasis was placed on Balanced Accuracy, Precision, Recall, F1-score, PR-AUC, and Matthews Correlation Coefficient. The obtained results indicate that the proposed model successfully balances the ability to detect proxy contracts while maintaining a relatively low number of false-positive predictions. In addition, the Average Precision obtained from the Precision–Recall analysis is considerably higher than the positive class prevalence, demonstrating that the classifier ranks proxy contracts substantially better than random guessing. These observations indicate that combining semantic code embeddings with handcrafted static Solidity features produces complementary information that improves the discrimination between proxy and non-proxy contracts, even under severe class imbalance. Similar observations have been reported in previous studies, where hybrid feature representations consistently improved classification robustness compared with relying solely on semantic or handcrafted features [23].

Despite the encouraging results, several limitations should be acknowledged. The experimental dataset consists of only 377 contracts, including 17 proxy contracts distributed across seven unique source-code groups. Although the proposed group-aware evaluation protocol improves the reliability of the reported performance, the limited number of positive samples may restrict the diversity of proxy implementations available for model learning. In addition, this study focuses exclusively on Ethereum smart contracts written in Solidity and formulates the problem as binary classification between proxy and non-proxy contracts. The proposed framework therefore does not distinguish among different proxy implementation patterns. Furthermore, the proposed approach relies solely on static source-code analysis using pretrained CodeBERT embeddings and handcrafted Solidity features without incorporating bytecode information or runtime execution behavior. Similar limitations have been identified in previous smart contract learning studies, where constrained dataset diversity and limited representation of minority classes affected model generalization [24].

Future work may extend this framework by utilizing larger and more diverse datasets, investigating multi-class proxy classification, and integrating additional static or dynamic features while maintaining the leakage-safe group-aware evaluation protocol to ensure reliable performance estimation. Moreover, evaluating recently developed pretrained code language models and graph-based program representations may further improve semantic understanding of proxy contract implementations while preserving the robustness of leakage-safe experimental evaluation [25].

Conclusion

This study proposed a leakage-safe framework for proxy smart contract detection by combining pretrained CodeBERT embeddings, handcrafted static Solidity features, and classical machine learning classifiers under a Stratified Group Cross Validation evaluation protocol. The proposed grouping strategy ensured that contracts sharing identical source code were assigned exclusively to either the training or testing partition, thereby preventing information leakage caused by duplicated contracts and providing a more realistic evaluation of model generalization.

Experimental results demonstrated that Logistic Regression achieved the best overall performance among the evaluated classifiers, outperforming Random Forest and XGBoost across most evaluation metrics. The findings indicate that pretrained CodeBERT embeddings provide meaningful semantic representations that, when combined with static source-code features, enable effective discrimination between proxy and non-proxy smart contracts, even under severe class imbalance. Furthermore, the use of group-aware evaluation revealed that reliable performance estimation is essential for datasets containing a high proportion of duplicated source code, as conventional evaluation protocols may produce overly optimistic results.

The primary contribution of this study lies in demonstrating that leakage-safe group-aware evaluation can be effectively integrated with pretrained code embeddings and classical machine learning to provide a reliable proxy smart contract detection framework. Beyond achieving competitive classification performance, the proposed methodology emphasizes the importance of preventing duplicate-code leakage when evaluating machine learning models for blockchain software analysis.

Although the proposed approach produced promising results, the study was limited by the relatively small number of proxy contracts and its focus on Ethereum Solidity source code. Future work may extend this research by constructing larger and more diverse datasets, investigating different proxy implementation patterns through multi-class classification, exploring additional static and dynamic features, and evaluating other pretrained code representation models while preserving the leakage-safe group-aware evaluation protocol.

Declarations

Author Contributions

Conceptualization: R.R., K.S.; Methodology: K.S.; Software: R.R.; Validation: R.R., K.S.; Formal Analysis: R.R., K.S.; Investigation: R.R.; Resources: K.S.; Data Curation: K.S.; Writing Original Draft Preparation: R.R., K.S.; Writing Review and Editing: K.S., R.R.; Visualization: R.R.; 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] Z. Zheng, S. Xie, H.-N. 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.

[2] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *Principles of Security and Trust*, M. Maffei and M. Ryan, Eds., Lecture Notes in Computer Science, vol. 10204. Berlin, Germany: Springer, vol. 2017, no. March, pp. 164–186, 2017, doi: 10.1007/978-3-662-54455-6_8.

[3] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: Automated vulnerability detection models for Ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2021, doi: 10.1109/TNSE.2020.2968505.

[4] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Stroudsburg, PA, USA: Association for Computational Linguistics, vol. 2020, no. November, pp. 1536–1547, 2020, doi: 10.18653/v1/2020.findings-emnlp.139.

[5] A. M. Ebrahimi, B. Adams, G. A. Oliva, and A. E. Hassan, "A large-scale exploratory study on the proxy pattern in Ethereum," *Empirical Software Engineering*, vol. 29, no. 4, Art. no. 81, 2024, doi: 10.1007/s10664-024-10485-1.

[6] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*, vol. 2020, no. October, pp. 530–541, 2020, doi: 10.1145/3377811.3380364.

[7] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, vol. 2018, no. May, pp. 9–16, 2018, doi: 10.1145/3194113.3194115.

[8] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, vol. 2016, no. October, pp. 254–269, 2016, doi: 10.1145/2976749.2978309.

[9] D. Guo *et al.*, "GraphCodeBERT: Pre-training code representations with data flow,"

in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=jLoC4ez43PZ

[10] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proceedings of the IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, vol. 2019, no. September, pp. 8–15, 2019, doi: 10.1109/WETSEB.2019.00008.

[11] Z. Liu, M. Jiang, S. Zhang, J. Zhang, and Y. Liu, "A smart contract vulnerability detection mechanism based on deep learning and expert rules," *IEEE Access*, vol. 11, no. July, pp. 77990–77999, 2023, doi: 10.1109/ACCESS.2023.3298048.

[12] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information Fusion*, vol. 81, no. May, pp. 84–90, 2022, doi: 10.1016/j.inffus.2021.11.011.

[13] R. Norvill, B. B. Fiz Pontiveros, R. State, and A. Cullen, "Standardising smart contracts: Automatically inferring ERC standards," in *Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, no. July, pp. 192–195, 2019, doi: 10.1109/BLOC.2019.8751350.

[14] Y. Huang, X. Wu, Q. Wang, Z. Qian, X. Chen, M. Tang, and Z. Zheng, "The sword of Damocles: Upgradeable smart contract in Ethereum," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension (ICPC)*, vol. 2024, no. June, pp. 333–345, 2024, doi: 10.1145/3643916.3644426.

[15] N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang, "Characterizing code clones in the Ethereum smart contract ecosystem," in *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, vol. 12059. Cham, Switzerland: Springer, vol. 2020, no. July, pp. 654–675, 2020, doi: 10.1007/978-3-030-51280-4_35.

[16] 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.

[17] 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.

[18] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers*, Dublin, Ireland, vol. 2022, no. May, pp. 7212–7225, 2022, doi: 10.18653/v1/2022.acl-long.499.

[19] F. Khan, I. Dávid, D. Varró, and S. McIntosh, "Code cloning in smart contracts on the Ethereum platform: An extended replication study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2006–2019, 2023, doi: 10.1109/TSE.2022.3207428.

[20] G. Varoquaux, "Cross-validation failure: Small sample sizes lead to large error bars," *NeuroImage*, vol. 180, pt. A, pp. 68–77, 2018, doi: 10.1016/j.neuroimage.2017.06.061.

[21] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Advances in Neural*

Information Processing Systems, vol. 35, 2022, pp. 507–520. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/hash/0378c7692da36807bdec87ab043cdadc-Abstract-Datasets_and_Benchmarks.html

[22] M. Kondo, G. A. Oliva, Z. M. Jiang, A. E. Hassan, and O. Mizuno, “Code cloning in smart contracts: A case study on verified contracts from the Ethereum blockchain platform,” *Empirical Software Engineering*, vol. 25, no. 6, pp. 4617–4675, 2020, doi: 10.1007/s10664-020-09852-5.

[23] S. Shakya, A. Mukherjee, R. Halder, A. Maiti, and S. Chattopadhyay, “SmartMixModel: Machine learning-based vulnerability detection of Solidity smart contracts,” in *Proceedings of the 2022 IEEE International Conference on Blockchain*, vol. 2022, no. September, pp. 37–44, 2022, doi: 10.1109/Blockchain55522.2022.00016.

[24] H. He and E. A. Garcia, “Learning from imbalanced data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009, doi: 10.1109/TKDE.2008.239.

[25] I. A. Qasse, M. Hamdaqa, and B. Þ. Jónsson, “PROXiFY: A bytecode analysis tool for detecting and classifying proxy contracts in Ethereum smart contracts,” in *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Seoul, South Korea, vol. 2025, no. January, pp. 4114–4117, 2025, doi: 10.1109/ASE63991.2025.00388.