Explainable Smart Contract Classification for Metaverse Applications Using Hybrid CodeBERT Embeddings and Static Code Analysis

P. Gajalakshmi[1,*], R. Elavarasi[2], D. Kadhiravan[3], M. Naresh[4]

[1,3,4]University College of Engineering, Tindivanam, India

[2]AMET Deemed to be University, Chennai, India

ABSTRACT

The rapid growth of blockchain technology has significantly increased the number of deployed smart contracts, particularly in metaverse ecosystems where they support applications such as non-fungible tokens (NFTs), decentralized finance (DeFi), tokens, decentralized autonomous organizations (DAOs), and digital marketplaces. The increasing volume and diversity of smart contracts make manual identification of their application categories inefficient and difficult to scale. Existing studies primarily focus on vulnerability detection, malware analysis, and code similarity, while research on automatic smart contract classification remains relatively limited. Furthermore, many deep learning approaches lack interpretability, making their prediction process difficult for blockchain developers to understand. This study proposes an explainable framework for automatic smart contract classification by combining CodeBERT embeddings, static Solidity code analysis, and machine learning. The proposed framework integrates semantic code representations with handcrafted structural features to construct a hybrid feature representation for classifying Solidity smart contracts into Token, NFT, DeFi, and other categories. Four machine learning algorithms, namely Random Forest, HistGradientBoosting, XGBoost, and Support Vector Machine (SVM), were evaluated using stratified cross-validation. Experimental results on a dataset of 377 Solidity smart contracts show that the Random Forest classifier achieved the best performance, obtaining an accuracy of 90.79%, macro precision of 87.59%, macro recall of 86.09%, macro F1-score of 85.81%, and macro-ROC-AUC of 98.51%. SHAP-based explainability analysis identified code_chars, code_tokens, n_mappings, n_functions, and n_requires as the most influential structural features affecting classification decisions. The results demonstrate that the proposed hybrid representation provides accurate and interpretable smart contract classification, offering a practical solution for supporting blockchain developers and researchers in managing the growing ecosystem of metaverse applications.

Keywords Smart Contract Classification, Metaverse, CodeBERT, Explainable Artificial Intelligence, Static Code Analysis

Submitted: 30 April 2026
Accepted: 12 August 2026
Published: 17 November 2026

Corresponding author
P. Gajalakshmi,
gajalakshmip23@gmail.com

Additional Information and Declarations can be found on page 304

DOI: 10.47738/ijrm.v3i4.88

Copyright
2026 Gajalakshmi et al.

Distributed under
Creative Commons CC-BY 4.0

INTRODUCTION

The rapid advancement of blockchain technology has accelerated the development of decentralized applications across various domains, particularly within the emerging metaverse ecosystem. Smart contracts serve as the fundamental building blocks of these applications by enabling autonomous execution of predefined agreements without relying on centralized intermediaries. In metaverse environments, smart contracts are extensively used to implement digital asset ownership through non-fungible tokens (NFTs), facilitate decentralized finance (DeFi) services, manage token economies,

How to cite this article: P. Gajalakshmi, R. Elavarasi, D. Kadhiravan, and M. Naresh, "Explainable Smart Contract Classification for Metaverse Applications Using Hybrid CodeBERT Embeddings and Static Code Analysis," *Int. J. Res. Metav.*, vol. 3, no. 4, pp. 289-307, 2026.

support decentralized autonomous organizations (DAOs), and operate virtual marketplaces [1]. As blockchain adoption continues to expand, the number of deployed smart contracts has grown rapidly, creating significant challenges in organizing, analyzing, and understanding their functionalities. Manual inspection of smart contracts has become increasingly inefficient because it requires substantial domain expertise and cannot scale to the continuously growing volume of blockchain applications.

To address these challenges, artificial intelligence has become an important approach for analyzing smart contract source code. Existing studies have primarily focused on security-related tasks, including vulnerability detection, malicious contract identification, clone detection, and source code similarity analysis [2]. These studies have demonstrated that machine learning and deep learning techniques can effectively learn semantic representations of source code and improve automated software analysis. More recently, pretrained programming language models such as CodeBERT have shown remarkable capability in capturing contextual information from source code, leading to significant improvements in software engineering tasks such as code search, code summarization, defect detection, and program classification [3]. Meanwhile, static code analysis remains widely adopted because it extracts interpretable structural characteristics, including function complexity, storage usage, control flow, and programming constructs, without executing the smart contract.

Although previous studies have demonstrated the effectiveness of CodeBERT and static code analysis for various software engineering applications, research specifically addressing automatic smart contract classification remains relatively limited. Most existing blockchain studies concentrate on identifying vulnerabilities, detecting malicious behaviors, or analyzing contract similarity, whereas comparatively little attention has been devoted to classifying smart contracts according to their functional roles within blockchain ecosystems. This limitation is particularly evident in metaverse applications, where smart contracts perform diverse functions such as token management, NFT ownership, decentralized finance, and digital asset trading. Furthermore, existing classification approaches frequently rely on either semantic representations or handcrafted software metrics individually, which may fail to capture both the contextual semantics and structural characteristics of Solidity source code simultaneously [4].

Another important limitation of existing approaches is the lack of model interpretability. Many recent deep learning models provide high predictive performance but operate as black boxes, making it difficult for blockchain developers, auditors, and researchers to understand why a particular smart contract is assigned to a specific application category. The absence of transparent decision-making reduces user confidence and limits the practical adoption of artificial intelligence for blockchain software analysis. Explainable Artificial Intelligence (XAI) techniques, such as SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME), have recently emerged as effective tools for interpreting machine learning predictions by identifying the features that contribute most significantly to classification outcomes. However, the application of explainability techniques to smart contract classification remains largely unexplored [5].

Motivated by these research gaps, this study proposes an Explainable Artificial

Intelligence framework for smart contract classification in metaverse applications using hybrid CodeBERT embeddings and static code analysis. The proposed framework combines semantic embeddings generated by CodeBERT with handcrafted static Solidity features to construct a hybrid feature representation capable of capturing both contextual and structural information from smart contracts. Four machine learning algorithms, namely Random Forest, HistGradientBoosting, XGBoost, and Support Vector Machine (SVM), are evaluated to identify the most effective classifier for this hybrid representation. To improve model transparency, SHAP is employed to explain the contribution of individual features and provide insights into the characteristics influencing classification decisions.

The main contributions of this study are threefold. First, a hybrid feature representation is proposed by integrating CodeBERT semantic embeddings with static Solidity code features for automatic smart contract classification. Second, a comprehensive comparative evaluation of multiple machine learning classifiers is conducted to identify the most effective model for categorizing metaverse-related smart contracts. Third, the proposed framework incorporates Explainable Artificial Intelligence through SHAP to provide interpretable classification results, enabling developers and researchers to better understand the structural characteristics that influence model predictions.

Literature Review and Related Works

Smart Contract Analysis

Smart contracts are self-executing programs deployed on blockchain platforms to automate transactions and enforce predefined agreements without relying on trusted intermediaries [6]. Their widespread adoption has enabled numerous decentralized applications, particularly in metaverse ecosystems where they facilitate digital asset ownership, decentralized finance, governance, and virtual marketplaces [7]. As the number of deployed smart contracts continues to increase, efficient automated analysis becomes increasingly important for supporting blockchain developers, auditors, and researchers.

Existing research on smart contract analysis has predominantly focused on security-related problems, including vulnerability detection, malicious contract identification, bytecode analysis, and code similarity assessment [8], [9], [10]. These approaches have significantly improved blockchain security by identifying software defects and potential attacks before deployment. However, comparatively fewer studies have investigated automatic classification of smart contracts according to their functional roles. Functional classification differs from vulnerability detection because the objective is to identify the application domain of a contract rather than detect implementation flaws. Such classification is particularly useful for blockchain indexing, repository organization, software reuse, and large-scale blockchain analytics [11].

Source Code Representation Using CodeBERT

Recent advances in deep learning have significantly improved source code representation through pretrained language models. Unlike traditional handcrafted software metrics, transformer-based models learn contextual semantic representations directly from programming languages using large-scale pretraining datasets [12]. Among these models, CodeBERT has become one of the most widely adopted pretrained models for software engineering

tasks because it jointly learns natural language and source code representations [13].

CodeBERT has demonstrated strong performance in several software engineering applications, including code search, code summarization, defect detection, clone detection, and program understanding [14]. The learned embeddings capture semantic relationships among programming constructs that are difficult to represent using conventional lexical features. Nevertheless, CodeBERT primarily models contextual semantics and does not explicitly describe structural characteristics of software, such as the number of functions, storage variables, mappings, or conditional statements. Consequently, relying solely on pretrained embeddings may overlook implementation-specific information that is particularly relevant in Solidity smart contracts [15].

Static Code Analysis for Smart Contracts

Static code analysis extracts structural characteristics from source code without executing the program, making it an efficient approach for analyzing large collections of smart contracts [16]. Common static features include the number of functions, mappings, modifiers, inheritance relationships, external calls, and conditional statements. These features provide useful indicators of software complexity, implementation style, and contract architecture.

In blockchain software engineering, static analysis has been widely applied for vulnerability detection and software quality assessment because it enables efficient examination of contract properties while avoiding execution overhead [17]. Although static features provide interpretable descriptions of contract structures, they cannot fully capture semantic relationships among programming statements. Therefore, combining static features with semantic representations may provide a more comprehensive characterization of Solidity source code than either representation alone.

Machine Learning for Smart Contract Classification

Machine learning techniques have become increasingly popular for software classification because they can automatically identify complex patterns from high-dimensional feature spaces [18]. Ensemble learning algorithms, including Random Forest, Gradient Boosting, and Extreme Gradient Boosting, have demonstrated robust performance across various software engineering applications due to their ability to model nonlinear feature interactions while maintaining good generalization capability. Support Vector Machines have also been widely used because of their effectiveness in handling high-dimensional datasets [19].

Although these algorithms have been extensively evaluated for software engineering tasks, relatively few studies have systematically compared their performance for functional smart contract classification using hybrid semantic and structural representations. Such comparative evaluation is important because different classifiers exhibit different learning characteristics when processing heterogeneous feature spaces that combine pretrained embeddings with handcrafted software metrics.

Explainable Artificial Intelligence

As machine learning models become increasingly complex, improving model interpretability has become an important research topic in artificial intelligence.

Explainable Artificial Intelligence (XAI) aims to provide transparent explanations for prediction results, allowing users to understand the factors influencing model decisions. Among the most widely adopted explanation techniques are SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME), which estimate the contribution of individual features to model predictions.

In software engineering, explainability has primarily been applied to defect prediction, vulnerability detection, and malware analysis [20]. However, its application to smart contract classification remains limited. Incorporating explainability into smart contract classification can improve user trust by revealing which semantic and structural characteristics influence the classification results. Such transparency is particularly valuable for blockchain developers and auditors who require understandable decision support when analyzing large numbers of deployed smart contracts.

Methodology

Research Framework

This study proposes an explainable machine learning framework for automatically classifying Solidity smart contracts into metaverse-related application categories. The proposed framework integrates semantic code representations generated by CodeBERT with handcrafted static Solidity features to construct a hybrid feature representation. The hybrid features are subsequently used to train several machine learning classifiers, while Explainable Artificial Intelligence techniques are employed to interpret the classification results. The overall research framework is illustrated in figure 1.

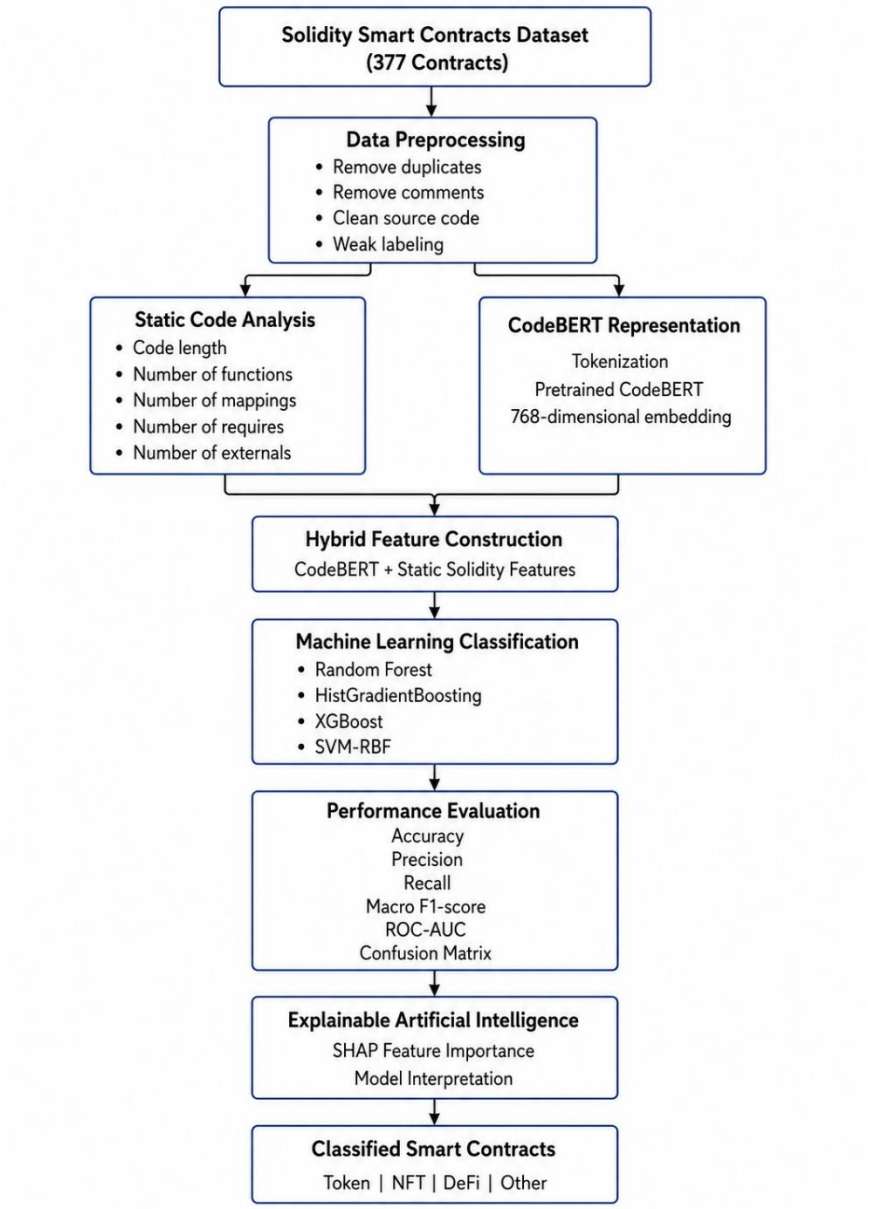


Figure 1 Research Framework

Dataset Preparation

The experimental dataset consists of 377 Solidity smart contracts collected from Ethereum-based blockchain applications. Each smart contract is represented by its Solidity source code and categorized into one of four metaverse-related application classes: Token, NFT, DeFi, and Other.

The dataset was cleaned before feature extraction to remove duplicated contracts, empty files, and unnecessary formatting. Comments, redundant whitespace, and non-essential characters were removed while preserving the original program structure. This preprocessing step ensures that both semantic embeddings and handcrafted static features are generated from consistent source code representations.

Since publicly available labeled datasets for smart contract functionality remain

limited, contract categories were generated using a weak labeling strategy based on Solidity-specific implementation patterns. Contract labels were assigned according to predefined rules that identify common programming constructs associated with each application category, such as ERC-20 interfaces for Token contracts and ERC-721 interfaces for NFT contracts. Although this approach enables efficient dataset construction, it may introduce labeling inaccuracies for contracts implementing multiple functionalities.

Static Solidity Feature Extraction

To capture structural characteristics of Solidity source code, a static analysis module was developed to extract software engineering features without executing the smart contracts.

The extracted features include source code statistics, structural complexity, storage-related characteristics, and programming constructs commonly used in Solidity. Examples include:

number of functions (n_functions)

number of external functions (n_external)

number of mappings (n_mappings)

number of require statements (n_requires)

number of modifiers

inheritance depth

number of events

code length

number of source code tokens

number of lines of code

These features describe the structural properties of each contract and provide interpretable information that complements semantic embeddings.

Before model training, numerical features were normalized to reduce differences in feature scales and improve classifier stability.

CodeBERT Embedding Generation

In addition to handcrafted features, semantic representations were generated using the pretrained CodeBERT model.

Each Solidity smart contract was tokenized using the CodeBERT tokenizer and processed by the pretrained transformer model to obtain contextual embeddings representing the semantic information contained in the source code. The final embedding vector consists of 768 dimensions, which encode contextual relationships among programming language tokens.

Unlike traditional lexical representations such as TF-IDF, CodeBERT captures programming semantics learned from large-scale source code corpora, enabling contracts with similar functionality to obtain similar vector representations despite differences in coding style.

Because CodeBERT embeddings were computationally expensive to generate, the extracted embeddings were cached and reused during subsequent

experiments.

Hybrid Feature Construction

The proposed framework combines semantic embeddings with handcrafted static features to construct a hybrid feature representation.

Let:

$$\mathbf{x}_{codebert} \in \mathbb{R}^{768} \quad (1)$$

denote the CodeBERT embedding vector and

$$\mathbf{x}_{static} \in \mathbb{R}^{m} \quad (2)$$

represent the extracted static Solidity features.

The hybrid representation is defined as:

$$\mathbf{x}_{hybrid} = [\mathbf{x}_{codebert}, \mathbf{x}_{static}] \quad (3)$$

the two feature vectors are concatenated into a single representation before classification.

This hybrid representation enables the classifier to simultaneously exploit semantic information learned by CodeBERT and structural characteristics extracted through static code analysis.

Smart Contract Classification

The hybrid feature vectors were used to train four supervised machine learning classifiers:

Random Forest

HistGradientBoosting

XGBoost

Support Vector Machine with Radial Basis Function (SVM-RBF)

To obtain reliable performance estimates, model evaluation employed Stratified K-Fold Cross Validation, ensuring that the class distribution was preserved across all folds.

The classifier with the highest Macro F1-score during cross-validation was selected for final evaluation on the independent test dataset.

Performance Evaluation

The proposed framework was evaluated using multiple performance metrics suitable for multi-class classification.

Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

Precision is computed as:

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \quad (6)$$

The harmonic mean of Precision and Recall is calculated using

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \quad (7)$$

To reduce the influence of class imbalance, macro-averaged Precision, Recall, and F1-score were adopted.

Additionally, classifier discrimination capability was evaluated using the one-versus-rest ROC curve and Macro ROC-AUC.

Explainable Artificial Intelligence

To improve model transparency, Explainable Artificial Intelligence was incorporated into the proposed framework.

The best-performing classifier was analyzed using SHapley Additive exPlanations (SHAP) to estimate the contribution of individual features to prediction outcomes. SHAP computes feature importance based on Shapley values derived from cooperative game theory, allowing each feature to be assigned a quantitative contribution to the model prediction.

Global explanations were generated to identify the most influential static Solidity features affecting classification decisions. The resulting feature importance rankings were subsequently compared with the intrinsic feature importance produced by the Random Forest classifier to evaluate the consistency of the explanations.

Results

Dataset Characteristics

The experimental dataset consists of 377 Solidity smart contracts collected from Ethereum-based applications and categorized into four metaverse-related application classes, namely Token, NFT, DeFi, and Other. These categories were selected to represent common functionalities within metaverse ecosystems, where smart contracts support digital asset management, decentralized finance, and virtual economic activities. Figure 2 illustrates the distribution of the collected smart contracts across the four categories. The dataset contains 220 Token contracts, 103 Other contracts, 34 DeFi contracts, and 20 NFT contracts, providing a diverse collection of contracts with varying functional characteristics and code complexity.

As shown in figure 2, the dataset exhibits a moderate class imbalance, with Token contracts accounting for 58.4% of the entire dataset, followed by other contracts (27.3%), DeFi contracts (9.0%), and NFT contracts (5.3%). Such an imbalanced distribution may bias conventional performance metrics, particularly overall classification accuracy, because the model can achieve high accuracy

by favoring the majority class. To provide a more reliable evaluation of classifier performance, this study employs macro-averaged Precision, Recall, and F1-score, which assign equal importance to each class regardless of its size. These metrics ensure that the classification performance of minority classes, especially NFT and DeFi contracts, is evaluated fairly alongside the majority classes.

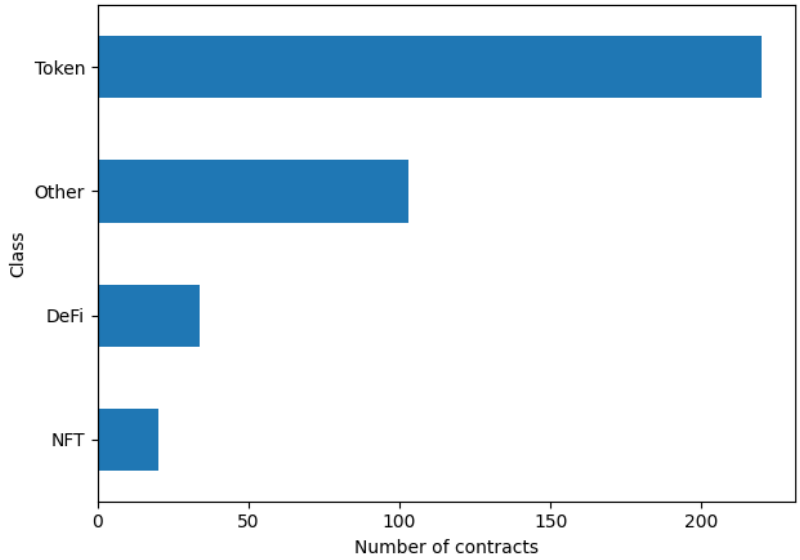


Figure 2 Distribution of smart contract classes used in the experiments.

CodeBERT Embedding Visualization

To examine the quality of the semantic representation learned by CodeBERT, the extracted embedding vectors were projected into a two-dimensional space using the t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm. Since CodeBERT generates a 768-dimensional embedding for each smart contract, direct visualization is not feasible. Therefore, t-SNE was employed to preserve the local neighborhood relationships among high-dimensional embeddings while reducing their dimensionality for visualization purposes. Figure 3 presents the resulting two-dimensional projection, where each point represents a Solidity smart contract and colors indicate the corresponding application category.

As illustrated in figure 3, smart contracts belonging to similar application categories tend to form local clusters in the embedding space, suggesting that CodeBERT captures meaningful semantic information from the source code. Token and Other contracts exhibit relatively distinguishable regions, whereas partial overlap is observed between the NFT and DeFi categories. This overlap is expected because these applications often share common Solidity constructs, including token standards, access control mechanisms, and transaction-related functions. Despite the presence of overlapping regions, the overall distribution demonstrates that the learned embeddings provide sufficient semantic separation among different contract categories, making them suitable as input features for the subsequent machine learning classifiers.

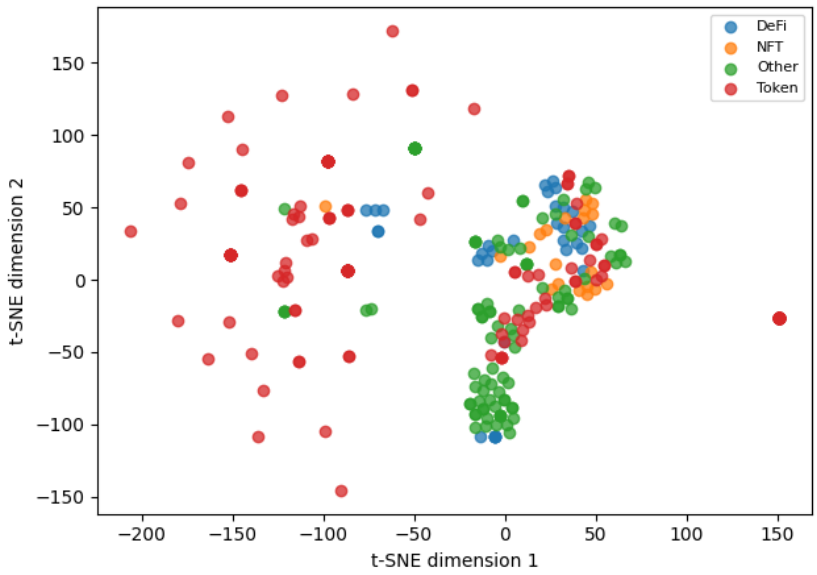


Figure 3 t-SNE visualization of CodeBERT embeddings.

Classification Performance

Four machine learning algorithms, namely Random Forest, HistGradientBoosting, XGBoost, and Support Vector Machine with a Radial Basis Function (SVM-RBF) kernel, were evaluated using stratified cross-validation to ensure that the class distribution was preserved across all validation folds. The evaluation was conducted using multiple performance metrics, including Accuracy, Macro Precision, Macro Recall, and Macro F1-score, to provide a comprehensive assessment of classifier performance on the imbalanced dataset. As presented in table 1, Random Forest achieved the highest overall performance with a Macro F1-score of 0.7126, followed closely by HistGradientBoosting (0.7125) and XGBoost (0.7069), while SVM-RBF obtained a substantially lower Macro F1-score (0.4952). Although Random Forest and XGBoost produced comparable classification accuracy, Random Forest demonstrated superior balanced performance across all classes, as reflected by its higher macro-averaged precision and F1-score. Based on these results, Random Forest was selected as the final classification model for subsequent testing and explainability analysis because it provided the best balance between predictive accuracy and class-wise classification performance.

Table 1 Cross-validation performance of the evaluated classifiers.

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1-score (Macro)
Random Forest	0.8570	0.7761	0.6997	0.7126
HistGradientBoosting	0.8505	0.7638	0.7053	0.7125
XGBoost	0.8572	0.7573	0.6988	0.7069
SVM-RBF	0.6645	0.5399	0.5940	0.4952

The Random Forest classifier, which achieved the best cross-validation performance, was subsequently evaluated on an independent test set to assess its generalization capability. The test results, summarized in table 2, demonstrate that the proposed hybrid feature representation combining

CodeBERT embeddings and static Solidity features achieved an overall accuracy of 90.79%, macro precision of 87.59%, macro recall of 86.09%, macro F1-score of 85.81%, and macro ROC-AUC of 98.51%. The high macro-averaged performance indicates that the classifier maintains consistent predictive capability across both majority and minority classes despite the inherent class imbalance in the dataset. Furthermore, the ROC-AUC value approaching 1.0 suggests excellent discriminative ability, indicating that the proposed hybrid representation effectively captures both the semantic and structural characteristics of Solidity smart contracts. These results confirm that the selected Random Forest model provides robust and reliable performance for automatically classifying smart contracts into metaverse-related application categories.

Table 2 Performance of the selected Random Forest model on the test dataset.

Metric	Value
Accuracy	0.9079
Precision (Macro)	0.8759
Recall (Macro)	0.8609
Macro F1-score	0.8581
ROC-AUC (Macro)	0.9851

Confusion Matrix Analysis

To further analyze the classification performance of the proposed model for each application category, a normalized confusion matrix was generated for the selected Random Forest classifier, as shown in figure 4. The confusion matrix provides a detailed view of the prediction results by comparing the true labels with the predicted labels, thereby highlighting the strengths and weaknesses of the classifier across different smart contract categories. Unlike aggregate evaluation metrics such as accuracy and macro F1-score, the confusion matrix enables a class-wise assessment of the model's ability to correctly distinguish between Token, NFT, DeFi, and Other smart contracts.

As illustrated in figure 4, the classifier correctly identified 93% of Token contracts, 90% of Other contracts, 86% of DeFi contracts, and 75% of NFT contracts. The highest classification performance was achieved for the Token and Other categories, indicating that these contracts exhibit distinctive semantic and structural characteristics that can be effectively captured by the proposed hybrid feature representation. In contrast, the NFT category produced the lowest recall, with most misclassified samples being assigned to other categories. This result is likely influenced by the relatively small number of NFT contracts available in the dataset and the similarity of certain implementation patterns shared with other smart contract types. Nevertheless, the confusion matrix demonstrates that the proposed model achieves consistent classification performance across all four categories while maintaining strong predictive capability for both majority and minority classes.

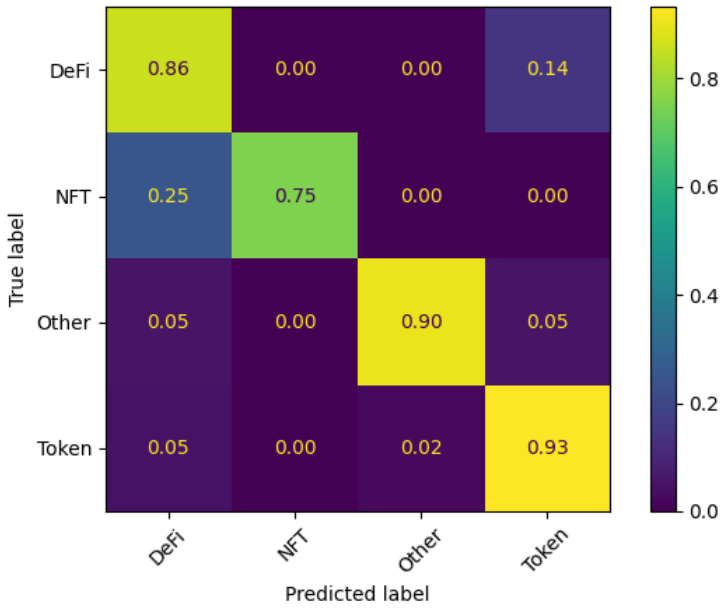


Figure 4 Normalized confusion matrix of the Random Forest classifier.

Explainable AI Analysis

To enhance the interpretability of the proposed smart contract classification framework, SHapley Additive exPlanations (SHAP) was employed to quantify the contribution of each static Solidity feature to the model's predictions. Unlike conventional feature importance measures, SHAP provides a theoretically grounded explanation by estimating the marginal contribution of individual features to the prediction outcome based on cooperative game theory. The resulting global feature importance is illustrated in figure 5, where features are ranked according to their average absolute SHAP values across all samples. This analysis enables a better understanding of how the Random Forest classifier utilizes structural characteristics of Solidity source code when distinguishing between different metaverse-related smart contract categories.

As shown in figure 5, the most influential features are code_chars, code_tokens, n_mappings, n_functions, and n_requires. The prominence of code_chars and code_tokens indicates that the overall size and complexity of the source code provide valuable information for differentiating contract types. Structural features such as n_mappings and n_functions reflect differences in storage organization and functional complexity among Token, NFT, DeFi, and Other contracts, while n_requires captures variations in conditional validation and access control logic commonly implemented in Solidity smart contracts. Furthermore, the ranking obtained from the SHAP analysis is consistent with the feature importance scores produced by the Random Forest model, suggesting that both explainability approaches identify similar structural characteristics as the primary factors influencing the classification decisions. This consistency increases confidence in the reliability and transparency of the proposed classification framework.

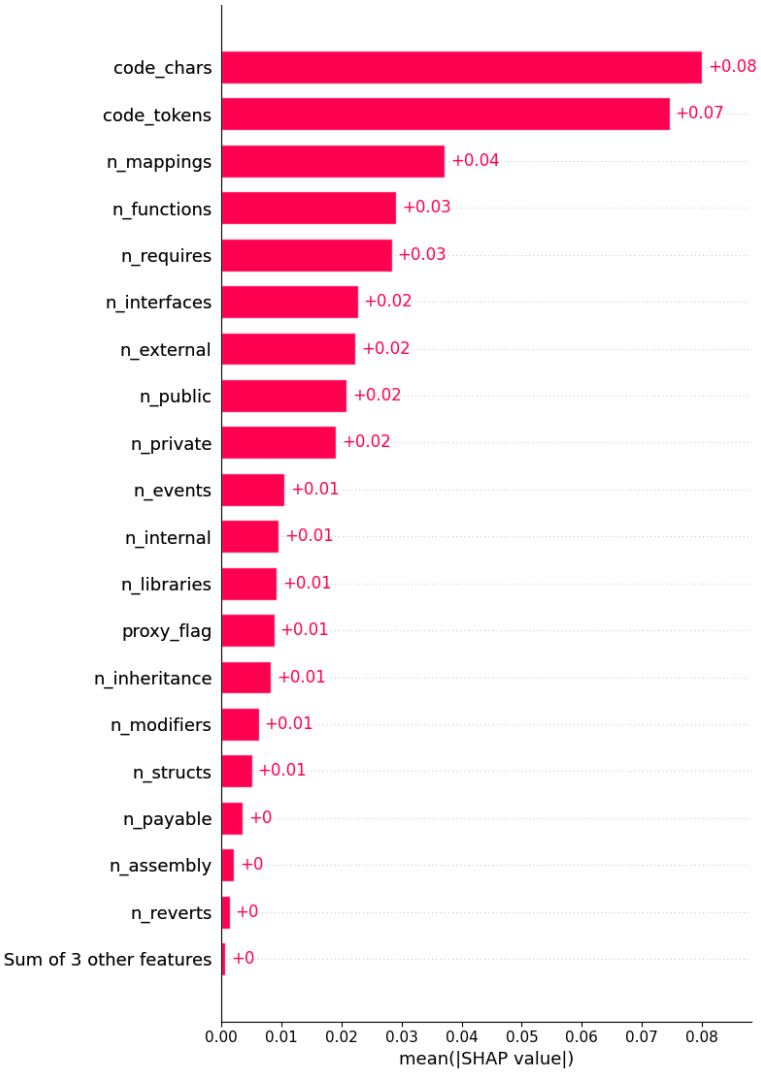


Figure 5 SHAP importance of static Solidity features.

Discussion

The experimental results demonstrate that the proposed hybrid framework provides an effective solution for the automatic classification of Solidity smart contracts in metaverse-related applications. By integrating semantic information extracted from CodeBERT embeddings with structural characteristics obtained through static Solidity code analysis, the proposed approach achieved an overall accuracy of 90.79%, a macro F1-score of 85.81%, and a macro ROC-AUC of 98.51%. These results indicate that the hybrid feature representation is capable of capturing complementary information that cannot be fully represented by either semantic embeddings or handcrafted static features alone [21]. CodeBERT encodes contextual and semantic relationships within the source code, allowing contracts with similar programming patterns to be represented in a meaningful feature space. However, semantic embeddings may not adequately describe implementation-specific characteristics such as storage organization, function complexity, or validation logic. The extracted static features address this limitation by explicitly describing structural properties of Solidity contracts, including the number of mappings, functions,

conditional statements, and external function calls [22]. The combination of these two feature types enables the classifier to learn richer representations of smart contracts, resulting in strong discriminative performance across Token, NFT, DeFi, and Other categories despite the imbalanced class distribution. Furthermore, among the evaluated machine learning algorithms, Random Forest consistently produced the best overall performance during cross-validation and testing. Its ensemble learning mechanism effectively models nonlinear relationships within heterogeneous feature spaces while maintaining robustness against noise and feature redundancy [23]. This finding provides a positive answer to RQ1 and RQ2, indicating that the proposed hybrid representation is effective for smart contract classification and that Random Forest is the most suitable classifier among the evaluated models.

The explainability analysis further demonstrates the practical value of the proposed framework by providing transparent insights into the model's decision-making process. The SHAP analysis identified code_chars, code_tokens, n_mappings, n_functions, and n_requires as the most influential features contributing to the classification results. These findings suggest that the overall size of the source code, the structural complexity of contract implementations, the organization of persistent storage through mappings, and the use of conditional validation logic are important indicators for distinguishing different smart contract application categories. The consistency between the SHAP rankings and the feature importance scores generated by the Random Forest classifier increases confidence that the model relies on meaningful software engineering characteristics rather than arbitrary statistical patterns [24]. Such interpretability is particularly important in blockchain applications, where developers and auditors require transparent explanations to understand and validate automated classification results. Nevertheless, several limitations should be acknowledged. The experimental dataset contains only 377 smart contracts, with a noticeable imbalance among application categories, particularly for the NFT class, which may affect the model's ability to generalize to more diverse contract implementations. In addition, the use of a weak labeling strategy based on predefined Solidity patterns may introduce labeling inaccuracies for contracts implementing multiple functionalities. Future work should therefore expand the dataset using manually verified annotations, incorporate additional metaverse-related categories such as DAO, Marketplace, Wallet, Oracle, and Bridge, and investigate the integration of complementary information sources such as bytecode analysis, runtime behavior, and symbolic execution [25]. These improvements are expected to further enhance both the predictive performance and the applicability of the proposed explainable smart contract classification framework in real-world metaverse ecosystems.

Conclusion

This study proposed an explainable hybrid framework for the automatic classification of Solidity smart contracts into metaverse-related application categories by integrating CodeBERT embeddings, static Solidity code analysis, and machine learning. The proposed framework combines semantic representations learned from source code with handcrafted structural features to capture complementary characteristics of smart contracts. Experimental evaluation on a dataset of 377 Solidity smart contracts demonstrated that the Random Forest classifier achieved the best overall performance among the evaluated algorithms, obtaining an accuracy of 90.79%, a macro precision of

87.59%, a macro recall of 86.09%, a macro F1-score of 85.81%, and a macro ROC-AUC of 98.51%. These results indicate that the proposed hybrid feature representation is effective for distinguishing Token, NFT, DeFi, and Other smart contract categories despite the presence of class imbalance. Furthermore, the SHAP-based explainability analysis revealed that structural characteristics such as code_chars, code_tokens, n_mappings, n_functions, and n_requires are the most influential features affecting the classification decisions, thereby improving the transparency and interpretability of the proposed framework.

The findings of this study demonstrate that combining semantic code embeddings with static code analysis provides an effective and interpretable approach for smart contract classification in metaverse ecosystems. In addition to achieving high predictive performance, the proposed framework offers practical value by providing explainable classification results that can assist blockchain developers, auditors, and researchers in understanding the characteristics of different smart contract applications. Nevertheless, several limitations remain, including the relatively small and imbalanced dataset, the use of weak labeling for contract annotation, and the focus on source code analysis without considering runtime behavior or bytecode information. Future research should investigate larger manually annotated datasets, include additional metaverse-related categories such as DAO, Marketplace, Wallet, Oracle, and Bridge, and explore the integration of dynamic analysis techniques and advanced code representation models. These improvements are expected to further enhance the robustness, generalizability, and practical applicability of explainable smart contract classification systems for emerging blockchain and metaverse applications.

Declarations

Author Contributions

Conceptualization: P.G., R.E., D.K., M.N.; Methodology: R.E.; Software: P.G.; Validation: P.G., R.E., D.K., M.N.; Formal Analysis: P.G., R.E., D.K., M.N.; Investigation: P.G.; Resources: R.E.; Data Curation: R.E.; Writing Original Draft Preparation: P.G., R.E., D.K., M.N.; Writing Review and Editing: R.E., P.G., D.K., M.N.; Visualization: P.G.; 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] S. Mystakidis, “Metaverse,” *Encyclopedia*, vol. 2, no. 1, pp. 486–497, 2022, doi: 10.3390/encyclopedia2010031.

[2] Y. K. Dwivedi et al., “Metaverse beyond the hype: Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy,” *International Journal of Information Management*, vol. 66, no. October, Art. no. 102542, pp. 1-55, 2022, doi: 10.1016/j.ijinfomgt.2022.102542.

[3] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “A survey on Ethereum systems security: Vulnerabilities, attacks, and defenses,” *ACM Computing Surveys*, vol. 53, no. 3, Art. no. 67, pp. 1–43, 2020, doi: 10.1145/3391195.

[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*, Online, vol. 2020, no. November, pp. 1536–1547, 2020, doi: 10.18653/v1/2020.findings-emnlp.139.

[5] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, “From local explanations to global understanding with explainable AI for trees,” *Nature Machine Intelligence*, vol. 2, no. 1, pp. 56–67, 2020, doi: 10.1038/s42256-019-0138-9.

[6] N. Szabo, “Formalizing and securing relationships on public networks,” *First Monday*, vol. 2, no. 9, 1997, doi: 10.5210/fm.v2i9.548.

[7] 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.

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

[9] 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 (CCS ’16)*, Vienna, Austria, vol. 2016, no. October, pp. 254–269, 2016, doi: 10.1145/2976749.2978309.

[10] 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 IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Gothenburg, Sweden, vol. 2018, no. May, pp. 9–16, 2018, doi: 10.1145/3194113.3194115.

[11] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, “Defining smart contract defects on Ethereum,” *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 327–345, 2022, doi: 10.1109/TSE.2020.2989002.

[12] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A survey of machine learning for big code and naturalness,” *ACM Computing Surveys*, vol. 51, no. 4, Art. no. 81,

pp. 1–37, 2018, doi: 10.1145/3233547.

[13] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, “CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online and Punta Cana, Dominican Republic, Nov. 2021, vol. 2021, no. November, pp. 8696–8708, doi: 10.18653/v1/2021.emnlp-main.685.

[14] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, “Unified pre-training for program understanding and generation,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, Online, Jun. 2021, vol. 2021, no. June, pp. 2655–2668, doi: 10.18653/v1/2021.naacl-main.211.

[15] M. R. I. Rabin, A. Mukherjee, O. Gnawali, and M. A. Alipour, “Towards demystifying dimensions of source code embeddings,” in *Proceedings of the 1st ACM SIGSOFT International Workshop on Representation Learning for Software Engineering and Program Languages*, Virtual Event, USA, Nov. 2020, vol. 2020, no. November, pp. 29–38, doi: 10.1145/3416506.3423580.

[16] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, “Securify: Practical security analysis of smart contracts,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Toronto, ON, Canada, Oct. 2018, vol. 2018, no. October, pp. 67–82, doi: 10.1145/3243734.3243780.

[17] 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 42nd IEEE/ACM International Conference on Software Engineering (ICSE)*, Seoul, South Korea, Jun.–Jul. 2020, vol. 2020, no. June, pp. 530–541, doi: 10.1145/3377811.3380364.

[18] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008, doi: 10.1109/TSE.2008.35.

[19] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, Aug. 2016, vol. 2016, no. August, pp. 785–794, doi: 10.1145/2939672.2939785.

[20] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, “Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Information Fusion*, vol. 58, no. June, pp. 82–115, 2020, doi: 10.1016/j.inffus.2019.12.012.

[21] Z. Cai, L. Lu, and S. Qiu, “An abstract syntax tree encoding method for cross-project defect prediction,” *IEEE Access*, vol. 7, no. November, pp. 170844–170853, 2019, doi: 10.1109/ACCESS.2019.2953696.

[22] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, “MadMax: Surviving out-of-gas conditions in Ethereum smart contracts,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, Art. no. 116, pp. 1–27, 2018, doi: 10.1145/3276486.

[23] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[24] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?: Explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’16)*, San Francisco, CA, USA, 2016, vol. 2016, no. August, pp. 1135–1144, doi: 10.1145/2939672.2939778.

[25] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding the greedy, prodigal, and suicidal contracts at scale,” in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC ’18)*, San Juan, PR, USA, 2018, vol. 2018, no. December, pp. 653–663, doi: 10.1145/3274694.3274743.