A Comparative Analysis of Hybrid Deep Learning Models for Reentrancy Vulnerability Detection in Ethereum Smart Contracts

Ahsan Adeleke Akoshile
ahsan.a.akoshile@stu.mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

Olamide Jogunola*
o.jogunola@mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

Mohammad Hammoudeh
m.hammoudeh@mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

Tooska Dargahi
t.dargahi@mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

ABSTRACT

Recent research has exposed significant security vulnerabilities within smart contracts that run on blockchain. Threats, such as, reentrancy attacks, where malicious actors exploit recursive function calls in a smart contract, pose a critical threat. This led to substantial financial losses in organisations. Traditional vulnerability detection methods, largely based on static analysis, showed limitations in effectively identifying reentrancy issues, often yielding high false positive rates and missing complex execution paths. This paper analyses hybrid deep learning models for reentrancy vulnerability detection in Ethereum smart contracts, introducing a unique approach that combines semantic and syntactic feature extraction. Specifically, our approach integrates CodeBERT embeddings for deep semantic insights with pattern-based feature vectors that capture Solidity constructs that are vulnerable to reentrancy attacks. Five hybrid models are evaluated, each selected to provide insights into structural and sequential dependencies within code. Findings highlighted the novelty of using multimodal feature integration in vulnerability detection, with models like Autoencoder-LSTM and CodeBERT-Transformer Encoder achieving high accuracy of 98.3% and 98.01%, respectively, demonstrating the effectiveness of hybrid architectures for capturing complex vulnerability patterns. This comparative study advances the smart contract security field, showcasing each model's strengths and trade-offs, and providing practical guidance for deploying deep learning-based vulnerability detection within blockchain ecosystems.

KEYWORDS

Deep learning, Smart contracts, Recurrent neural networks, Feature extraction, Convolutional neural networks, Security, Solidity

ACM Reference Format:
Ahsan Adeleke Akoshile, Olamide Jogunola, Mohammad Hammoudeh, and Tooska Dargahi. 2024. A Comparative Analysis of Hybrid Deep Learning Models for Reentrancy Vulnerability Detection in Ethereum Smart Contracts. In *The 8th International Conference on Future Networks & Distributed Systems (ICFNDS '24), December 11–12, 2024, Marakech, Morocco.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3726122.3726256

ICFNDS '24, December 11–12, 2024, Marakech, Morocco

ACM ISBN 979-8-4007-1170-1/24/12.
https://doi.org/10.1145/3726122.3726256

1 INTRODUCTION

Blockchain technology revolutionised various industries by introducing decentralised, secure, and transparent systems for digital transactions and data management. Originally proposed as the underlying technology for Bitcoin by Satoshi Nakamoto in 2008, blockchain was designed to enable trustless peer-to-peer transactions without centralised intermediaries, thereby enhancing security and reducing dependency on traditional financial institutions. This decentralised ledger technology gained further traction with the launch of Ethereum in 2015, which extended blockchain capabilities by introducing programmable smart contracts, these are self-executing contracts with predefined rules embedded in code [3]. Unlike Bitcoin, Ethereum's blockchain was developed as a platform for decentralised applications (dApps), such as finance, supply chain, healthcare, and gaming. However, as adoption has grown, so too have the challenges associated with ensuring the security and reliability of blockchain and smart contracts [12].

Among these challenges, reentrancy vulnerabilities are particularly critical due to their potential to inflict severe financial damage by allowing attackers to repeatedly invoke contract functions before completing initial executions [11, 13, 27]. The infamous DAO attack in 2016 is one of the most severe examples of reentrancy exploitation, resulting in the loss of approximately 3.6 million ETH (around 60 million USD at the time) by repeatedly calling a vulnerable function within the contract before it updated its state [15]. In 2020, the dForce reentrancy vulnerability attack resulted in a loss of 25 million USD from the Lendf.me protocol. This reveals vulnerabilities within the decentralised finance (DeFi) space [10]. Instances such as these highlight the need for techniques to detect and mitigate vulnerabilities with smart contracts.

Traditional techniques rely on static analysis and smart contract code examination based on established patterns or rules [1, 20]. These techniques identifies certain types of smart contract vulnerabilities at a structural level. They are inadequate in identifying more complex issues, especially vulnerabilities that only emerge during runtime, such as reentrancy. The capability to simulate the dynamic behaviours of smart contracts is lacking in static analysis,

resulting in difficulty to capture vulnerabilities on specific execution paths [4]. However, these tools can efficiently flag potential risks, but are often associated with producing high negative rates or false positive. This could result in either overestimation or underestimation of security threats. In addition, to address the dynamic nature and evolving complexity of smart contract vulnerabilities, more advanced solutions are required.

Recently, research effort on detecting anomalies on blockchain network [18], and detecting vulnerabilities in smart contracts [2] have focused on the use of machine learning (ML) and deep learning (DL) techniques. ML approaches such as Random Forests, Support Vector Machines (SVMs), and Gradient Boosting, have classified smart contracts as vulnerable or safe leveraging the handcrafted features derived from the code, such as opcode sequences or dependency graphs [2]. However, complex, non-linear relationships inherent in smart contract vulnerabilities are often not captured by these methods due to their reliance on feature engineering. DL-based approaches have shifted towards automated feature extraction, such as the use of Convolutional Neural Networks (CNNs) to detect anomalies and structural patterns in contract code [27]. Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) and Bidirectional Gated Recurrent Units (BiGRUs), usually models the sequential dependencies in contract code, for a deeper understanding of execution flows which is critical for identifying vulnerabilities such as reentrancy attacks [26]. Likewise, a semantic layer that incorporates both natural and programming language representations is introduced by transformer-based architectures such as CodeBERT. This significantly improved the detection of relationships and intricate patterns in smart contracts [7]. Despite their effectiveness, the dependency on extensive, accurately labeled datasets and the potential for overfitting to specific patterns are some limitations of these DL models. These limitations provide opportunities for refinement, in improving generalisability and enhancing the efficiency of models across diverse smart contract datasets.

To solve these challenges, hybrid ML and DL architectures have emerged as a promising solution, leveraging the strengths of different models to capture both the structural and sequential complexities in smart contract code. By combining different DLs, such as CNNs and RNNs, and LSTM and BiGRUs, these models can learn from complex patterns within code sequences that traditional static analysis tools are likely to miss. In addition, the integration of advanced transformer-based embeddings, such as CodeBERT, enables the incorporation of semantic understanding within the models, providing a contextually rich layer enhancement to identify vulnerabilities associated with reentrancy. By combining these architectures, each model offers a multidimensional approach to vulnerability detection, addressing limitations of the models in isolation and improving both the accuracy and precision of smart contract security assessments.

In detecting reentrancy vulnerabilities in Ethereum smart contracts, this study evaluates the performances of five distinct hybrid deep learning models. These include, CNN-BiGRU, Autoencoder-LSTM, CodeBERT-CNN-BiLSTM, CodeBERT-Transformer Encoder, and CodeBERT-GRU. To sum, the specific contributions are as follows:

- Development of five hybrid deep learning models for the detection of reentrancy vulnerabilities in smart contracts on Ethereum blockchain network. Each of the deep learning architecture are uniquely selected and combined to capture specific aspects of code structure and behaviour to identify nuanced vulnerability patterns.
- Enhancing the performance of these hybrid models by performing robust data preprocessing methods including the use of semantic embedding extraction using CodeBERT to capture the rich semantic content and pattern-based feature engineering to identify specific constructs within the smart contracts.
- Evaluating the models based on several key metrics, highlighting their usefulness, outcome and trade-offs, whilst providing recommendations based on their performances.

By conducting a detailed comparison across key metrics, this paper seeks to provide insights into the effectiveness of each model, offering a clearer understanding of the potential and limitations of deep learning for advancing smart contract security.

The remaining sections of this paper are organised as follows: Section 2 presents the architecture of the evaluated hybrid deep learning models. Section 3 outlines the methodology and experimental setup, providing details on the dataset composition and preprocessing pipeline. It also describes the evaluation metrics utilised to assess the effectiveness of the models in identifying reentrancy vulnerabilities. The results are presented in Section 4. This section discusses the performance of the comparative analysis of the five hybrid models using the key metrics discussed in Section 3, highlighting each model's strengths and limitations in the context of vulnerability detection. Finally, the conclusion is presented in Section 5 providing the summary of the findings and recommendations for future research aimed at advancing smart contract security through DL innovations.

2 HYBRID LEARNING MODELS FOR REENTRANCY DETECTION

Reentrancy vulnerabilities is illustrated in Fig. 1. This occurs when an external call to another contract is made before updating the internal state of a smart contract, allowing attackers to repeatedly re-enter the function and manipulate the contract's logic [25]. Functions such as call.value, delegatecall, or fallback are some of the targets being exploited by threat actors, enabling recursive calls that drain funds or cause inconsistent state changes. To address the complexity of reentrancy vulnerability detection in Ethereum smart contracts using DL, in this section, five distinct hybrid DL models are discussed.

2.1 CNN-BiGRU with Attention Model

The CNN-BiGRU with Attention model is effective in capturing both spatial and sequential features in smart contract code. CNNs are known to identify local dependencies and spatial hierarchies in code, making them suitable for analysing the syntactic structure in code sequences [14]. Thus, the CNN component in this hybrid model is used to extract spatial patterns from the code, particularly relevant for recognising structural code features. Following the CNN layers, the model employs BiGRU, enabling it to process the code sequence

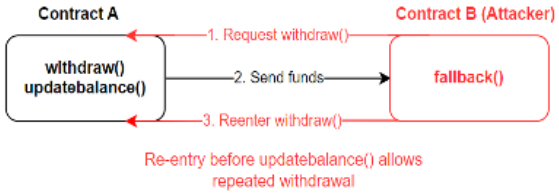


Figure 1: An illustration of a reentrancy attack on a smart contract.

bidirectionally capturing dependencies across both past and future contexts in the contract. This bidirectional approach is crucial for modeling the long-term dependencies that could signal reentrancy vulnerabilities or other logical issues. Studies have shown that BiGRU, when combined with CNN, can enhance the ability of the model to learn both local and global patterns within code, which is crucial for detecting complex vulnerabilities [21, 26].

The attention mechanism in the hybrid model is used to refine the model's focus within the sequence and is integrated with the BiGRU layer. This mechanism assigns weights to different parts of the sequence, enabling the model to focus on critical sections of the code that are more likely to contain vulnerabilities. Attention mechanisms have proven highly effective in vulnerability detection tasks, as they allow the model to dynamically allocate resources to sections of code with high-risk constructs, such as call.value or delegatecall functions commonly associated with reentrancy risks [14]. The combined CNN-BiGRU with Attention model offers a balanced approach to capture both the structural and semantic patterns in smart contract code, resulting in a model that is well-suited for precise detection of reentrancy vulnerability.

2.2 Autoencoder-LSTM Model

Both autoencoder and LSTM architectures are combined in the hybrid Autoencoder-LSTM model for reentrancy vulnerability detection in smart contract. This model is effective in reducing the dimensionality of the input data while capturing temporal patterns essential for analysing sequential dependencies in smart contracts. Specifically, autoencoders compress the contract representations by encoding input data into a reduced-dimensional space, thereby retaining only the essential features relevant for detecting vulnerabilities. The resulting lower-dimensional embedding is further processed by the LSTM layers, which are well-suited for capturing temporal relationships in sequential data. Studies, such as [16], have demonstrated that LSTM-based models can capture sequential dependencies in code, which is especially valuable for modeling smart contract behaviour to identify reentrancy vulnerabilities.

The ability to process and reconstruct sequential data efficiently, has made Autoencoder-LSTM model widely adopted in detecting various types of anomalies and vulnerabilities across different domains [22]. For instance, in network security, autoencoder-LSTM architectures have been successfully employed to distinguish between normal and malicious patterns by analysing the reconstruction error across input sequences [17]. The autoencoder compresses the input, while the LSTM identifies sequence patterns, which enable the model to detect deviations that may be an indication of vulnerabilities, such as unchecked state updates or improper use of external calls in smart contracts. In this study, the Autoencoder-LSTM model contributes to the detection of reentrancy vulnerabilities by allowing the model to process both the compressed and sequentially contextualised data from the smart contracts, which are essential for identifying complex vulnerabilities in decentralised systems.

2.3 CodeBERT-CNN-BiLSTM Model

The CodeBERT-CNN-BiLSTM model combines CodeBERT embeddings with CNN and BiLSTM layers. This combination enhances the model to capture both semantic and structural information in smart contract code, reflecting its effectiveness in detecting reentrancy vulnerabilities. In smart contract vulnerabilities detection, understanding context and interdependencies between code sections is essential for identifying risks associated with external calls and state changes. CodeBERT is well-suited for this task. It is a transformer-based model that is trained on programming languages. It provides deep contextual embeddings by capturing semantic relationships and the functional intent of code segments. This is particularly valuable for understanding the intricate structures in smart contracts [7]. This embedding allows the model to interpret complex patterns and identify potential vulnerabilities, such as insecure calls and unchecked conditions that are common in reentrancy-prone contracts [28].

CNN layer follows the embedding layers and are applied to detect spatial features in the code. The CNNs are effective in recognising local patterns, making them suitable for identifying syntactic elements and structural patterns that signal potential vulnerabilities [14]. In this model, the CNN layers captures code-specific patterns, thereby enhancing the detection of structural irregularities that can contribute to security risks in smart contracts. To further improve vulnerability detection, BiLSTM layers are then applied after the CNN layers to capture temporal dependencies within the code, essential for understanding execution order and operational flow. The bidirectional nature of BiLSTM allows it to capture dependencies in both the forward and backward direction through code sequences, making it especially useful for identifying reentrancy vulnerabilities that depend on specific execution paths [25]. The nybrid CodeBERT-CNN-BiLSTM model provides a comprehensive approach for detecting multi-layered signals associated with reentrancy vulnerabilities in Ethereum smart contracts.

2.4 CodeBERT-Transformer Encoder Model

The CodeBERT-Transformer Encoder Model integrates CodeBERT embeddings with a transformer encoder architecture, thereby combining both semantic understanding with the ability to model complex relationships and dependencies within the smart contract code. The CodeBERT captures both syntactic and semantic nuances of code segments to represent the functional and structural context of smart contract code effectively [7]. The transformer encoder component further processes the embeddings from CodeBERT, leveraging multi-head self-attention mechanisms to capture intricate patterns and long-range dependencies in the code, examining different parts

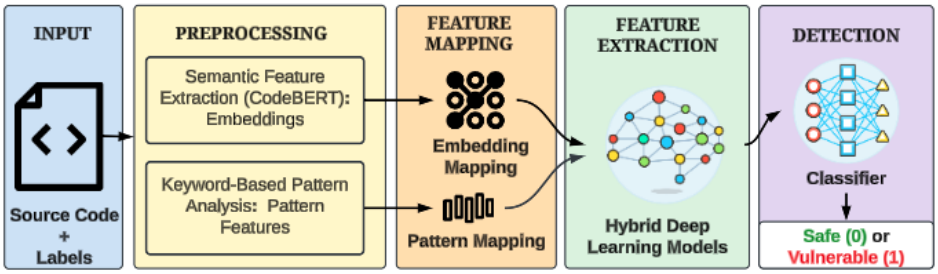


Figure 2: Flowchart illustration of the workflow architecture.

simultaneously. This is a critical feature for detecting vulnerabilities like reentrancy, which often depend on multi-step interactions across various contract functions. Research has demonstrated that transformer architectures can model complex control flows and data dependencies between various function calls, making them highly effective for vulnerability detection through capturing context and cross-token dependencies. This feature enables more accurate identification of code vulnerabilities compared to sequential models [8].

Study [5] argues that transformer-based approaches leverages semantic embeddings from models like CodeBERT to enhance the accuracy of vulnerability detection by capturing both structural and functional aspects of smart contracts. Ultimately, the model captures a broad spectrum of vulnerability indicators by integrating semantic embeddings with sophisticated attention mechanisms. This combination allows it to detect vulnerabilities that manifest in both the underlying logic and explicit code structure. This multi-layered capability positions the model as a strong candidate for reentrancy vulnerability detection in high-stakes environments like DeFi, where code reliability and security are paramount.

2.5 CodeBERT-GRU with Pattern Embeddings Model

In providing a comprehensive approach to smart contract vulnerability detection, the CodeBERT-GRU with Pattern Embeddings Model combines the semantic representation of CodeBERT with a GRU layer and an additional pattern-based embedding. CodeBERT provides the foundational embedding layer by capturing the contextual meaning and functional nuances of code. This helps in identifying complex vulnerability patterns through recognising both high-level code semantics and the structural flow needed to understand critical functions. While the GRU layer captures sequential dependencies across code segments, enabling the model to effectively process token sequences in smart contracts where order and function calls play a key role in detecting vulnerabilities. GRUs also captures dependencies in code sequences with reduced computational load compared to other recurrent models, making them efficient without sacrificing performance [23].

The pattern-based embedding layer of this hybrid model, encodes the frequency and presence of specific Solidity constructs commonly associated with reentrancy risks. This pattern-based feature engineering complements the deep embeddings from CodeBERT by adding a layer of vulnerability-specific insights, that flags reentrancy-prone constructs and functions in the smart contract code. The study by [28] showed the effectiveness of such targeted pattern embeddings in highlighting high-risk segments within contracts, further improving detection rates.

3 EVALUATION OF THE HYBRID MODELS

The methodology used in developing and evaluating the hybrid DL models for reentrancy vulnerability detection is described in this section. It also includes details on data collection, data preparation, preprocessing pipeline, model architectures, and evaluation metrics. Fig 2 illustrates the overall workflow flowchart of this work.

3.1 Data Collection and Dataset Composition

To build a high-quality and comprehensive dataset for reentrancy vulnerability detection, this study sourced

Two major repositories: the SmartBugs (SB) Curated Dataset [6] and the Messi-Q Smart Contract Dataset [19] are the sources of dataset used for this study. The SB Curated Dataset is an integral part of the SmartBugs framework that is widely recognised for its extensive vulnerability tagging based on the DASP (Decentralised Application Security Project) taxonomy. This dataset covers critical issues in smart contracts such as reentrancy, unchecked low-level calls, and access control weaknesses. These detailed annotations make it a fundamental resource in blockchain security research for benchmarking the comparative analysis of smart contract vulnerability detection tools. The Messi-Q Smart Contract Dataset complements this by offering a substantial collection of real-world Ethereum smart contracts. The size and diversity of this dataset, capturing a broad spectrum of contract structures and operational contexts makes it invaluable in security research on Ethereum blockchain. This enables the analysis of vulnerability patterns, especially the frequent use of Solidity's call.value and delegatecall functions, which are well-known indicators of reentrancy risks in smart contracts.

The collected smart contract dataset were manually reviewed to eliminate duplicates and retain only the core code, ensuring data integrity and relevance for model training. Ultimately, this dataset comprises 1,563 reentrancy-vulnerable contracts and 3,126 non-vulnerable contracts, totalling 4,689 contracts. This balanced 2:1 ratio provides a suitable foundation for training DL models, offering a well-rounded dataset that captures both real-world complexity and specific vulnerability indicators.

3.2 Data Preprocessing

The preprocessing pipeline uses CodeBERT with pattern-based feature engineering to combine semantic embedding extraction and capturing both the structural and contextual aspects essential for identifying vulnerability risks in Ethereum smart contracts.

3.2.1 Semantic Embedding Extraction Using CodeBERT. In our pipeline, we used CodeBERT's AutoTokeniser for tokenisation and AutoModel for embedding generation. The AutoTokeniser from CodeBERT was used to convert each Solidity contract into a sequence of tokens. Tokenisation is a critical step, as it breaks down the code into interpretable units while preserving context. Each sequence was configured to a maximum token length of 512, using the *max_length* parameter, which ensures uniformity across inputs by truncating longer sequences and padding shorter ones. This padding enables each contract to maintain context completeness within the model's capacity, eliminating computational inefficiencies that arise from variable-length sequences. Padding also supports batched processing by ensuring consistent input dimensions, crucial for large-scale embeddings.

After tokenisation, each tokenised sequence was passed through CodeBERT's AutoModel to produce contextual embeddings. AutoModel outputs high-dimensional vector representations by capturing relationships between tokens in each sequence, derived from the final hidden layer of the transformer. The embeddings generated are high-dimensional matrices (tensor representations), converted to fixed-size matrices using PyTorch tensor operations for consistency in downstream model processing. By preserving these hierarchical and bidirectional contexts, the output embeddings reflect both high-level code logic and intricate syntax.

3.2.2 Pattern-Based Feature Engineering . In addition to semantic embeddings, the preprocessing pipeline includes a pattern-based feature engineering step that identifies specific Solidity constructs known to increase susceptibility to reentrancy vulnerabilities. This targeted approach highlights syntactic indicators directly related to common reentrancy attack patterns, providing explicit cues that aid the model in recognising vulnerable code structures.

Six Solidity constructs; call, delegatecall, send, transfer, fallback, and modifier were identified as key indicators of reentrancy risks. These constructs facilitate external calls and state changes, which, if not adequately protected, can lead to exploitation through reentrancy. Specifically, functions like call.value and delegatecall are critical in reentrancy attacks because they allow external code execution within the calling contract's context. Unprotected usage of these functions can open vulnerabilities to reentrant calls, allowing repeated or recursive function calls before the initial execution completes [27].

To quantify these reentrancy-prone constructs, a custom feature extraction function was implemented to count occurrences of each target pattern within each contract. This process generates a fixed-length vector, with each position representing the frequency of a particular construct. This vectorised representation provides a syntactic footprint of each contract, capturing structural details that signal potential vulnerabilities. By storing this vector alongside CodeBERT embeddings, the model gains access to both implicit (contextual) and explicit (pattern-based) reentrancy indicators, creating a well-rounded feature set that enhances vulnerability detection.

3.2.3 Data Cleansing and Integrity. Following embedding extraction and feature engineering, a data cleansing process was applied to improve dataset relevance and integrity. Duplicates were removed to prevent redundancy, and non-essential elements such as comments and excessive whitespace were stripped. This streamlined each contract to its executable logic, ensuring that the model focuses exclusively on vulnerability-relevant patterns and reducing noise from extraneous content.

3.2.4 Feature Normalisation. The final preprocessing step involved feature normalisation, applied to both semantic embeddings and pattern-based feature vectors to ensure uniform scaling across all features. Normalisation is critical in deep learning, as it standardises feature values, preventing any single feature from disproportionately influencing the model during training and improving overall training stability [24]. For this study, we applied z-score normalisation, adjusting each feature by subtracting its mean μ and dividing by the standard deviation σ, as defined by $\frac{x-\mu}{\sigma}$.

This process results in each feature having a mean of zero and a standard deviation of one, standardising the scale across features. By applying z-score normalisation, we optimised the convergence rate of gradient-based optimisers, which benefit from consistent gradient updates across features. This regularity in feature scaling enhances model stability and improves learning efficiency, contributing to balanced training dynamics and a more effective overall learning process.

3.3 Simulation setup and Evaluation Metrics

In this study, model training and evaluation were carried out using Google Colab Pro [9], employing a TPU v2-8 configuration with 334.6 GB RAM to handle computationally intensive tasks such as the generation of CodeBERT embeddings and the training of deep learning models. This setup provided the necessary computational resources for managing the large and complex datasets characteristic of smart contract code analysis.

The dataset was split into 70% for training, 15% for validation, and 15% for testing, with a fixed $random\_state = 42$ to ensure consistent reproducibility of results across each split. Two distinct code implementations were used for data splitting to address different feature structures. The first split configuration was applied to multi-feature data scenarios, where multiple feature arrays, such as semantic embeddings and pattern-based feature vectors, were maintained in alignment across all data subsets. This ensured that training, validation, and testing sets contained matching rows for each feature array, thus preserving consistency across model inputs. In cases involving a single feature set and label array, a streamlined single-feature split code was used, simplifying the process for efficient handling when no additional feature sets were present.

The effectiveness of each model in detecting reentrancy vulnerabilities was evaluated using five core metrics: accuracy, precision, recall, F1-score, and ROC-AUC, each providing insights into different aspects of the model's performance.

Accuracy, calculated as the ratio of correctly predicted instances to the total predictions, offers an overview of overall model performance:

$$\mathrm{A} = \frac{TP + TN}{TN + FN + TP + FP} \tag{1}$$

where TN is true negatives, which is the number of correctly predicted vulnerabilities. FP is the false positives and depicts the number of non-vulnerabilities incorrectly predicted as vulnerabilities. FN is false negatives, which is the number of reentrancy vulnerabilities incorrectly predicted as non-vulnerabilities; TP is true positives that predicts the number of correctly predicted reentrancy vulnerabilities.

Precision, P, measures the proportion of true positive predictions out of all positive predictions, making it particularly relevant for assessing the model's ability to minimise false positives in a security-sensitive application. It is defined as:

$$\mathrm{P} = \frac{TP}{TP + FP} \tag{2}$$

Recall, R, or sensitivity, calculates the proportion of actual positive cases that are correctly identified by the model. This metric is crucial in vulnerability detection to minimise missed cases (false negatives):

$$\mathrm{R} = \frac{TP}{TP + FN} \tag{3}$$

The F1-score, which is the harmonic mean of precision and recall, offers a balanced measure that is particularly informative when both false positives and false negatives carry significant costs. The F1-score is given by:

$$\mathrm{F1} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

Finally, the ROC-AUC (Receiver Operating Characteristic Area Under the Curve) metric was used to evaluate the model's ability to distinguish between vulnerable and non-vulnerable classes across various threshold settings. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at different threshold levels, with the AUC providing a single value to assess the model's separability. The ROC-AUC is formally represented as:

$$\mathrm{ROC} = 1 - \frac{TN}{TN + FP} \tag{5}$$

Each of these metrics was chosen for its relevance to security-sensitive applications where classification accuracy, especially regarding false positives and negatives, is critical. The combination of these metrics offers a comprehensive view of model performance, ensuring that both general accuracy and nuanced vulnerability detection capabilities are accounted for in the evaluation.

4 RESULTS AND DISCUSSION

The discussed five deep learning models for detecting reentrancy vulnerabilities in Ethereum smart contracts are evaluated in this section using the defined key metrics. The models were evaluated using five core metrics: accuracy, precision, recall, F1-score, and ROC-AUC, and the results are discussed.

Table 1 presents the summary of the performance metrics for each model. This table shows that the Autoencoder-LSTM model achieved the highest accuracy (98.30%) and F1-score (97.20%), suggesting that its architecture is particularly effective at learning compact and informative representations, which may capture subtle patterns indicative of vulnerabilities. Similarly, the CodeBERT-Transformer Encoder model demonstrated resilient performance with an accuracy of 98.01% and an F1-score of 96.71%. This model's use of attention mechanisms appears to enhance its ability to capture complex contextual relationships in code, providing both high recall (94.50%) and high precision (99.04%). The CNN-BiGRU with Attention model also showed a strong balance in performance, with an accuracy of 94.74% and an F1-score of 91.21%, making it a viable option for initial vulnerability detection stages. The CodeBERT-CNN-BiLSTM model, while scoring the highest recall at 98.62%, showed a trade-off in precision 81.44%, resulting in more false positives. This characteristic could be advantageous in settings where detecting all potentially vulnerable contracts is prioritised, allowing a secondary model to refine results by filtering false positives. Finally, the CodeBERT-GRU model achieved competitive accuracy (95.17%) and balanced precision and recall, making it versatile for scenarios requiring balanced sensitivity and specificity.

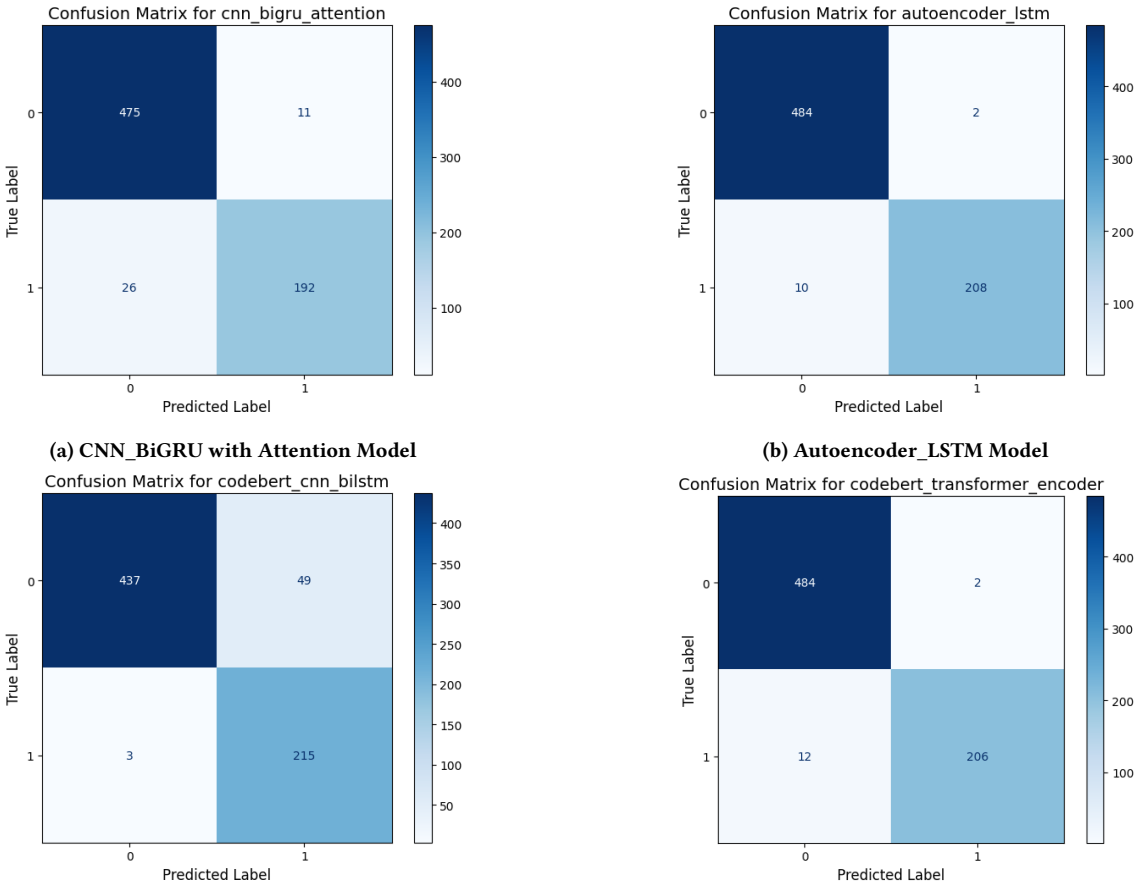
Table 1: Performance Metrics of all Models

Model	Accuracy	Precision	Recall	F1-Score
CNN-BiGRU-Attention	0.9474	0.9458	0.8807	0.9121
Autoencoder-LSTM	0.9830	0.9905	0.9541	0.9720
CodeBERT-CNN-BiLSTM	0.9261	0.8144	0.9862	0.8921
CodeBERT-Transform-Encoder	0.9801	0.9904	0.9450	0.9671
CodeBERT-GRU	0.9517	0.9340	0.9083	0.9209

To further interpret these results, confusion matrices in Fig. 3 were generated for each model, providing insight into their handling of true positives, true negatives, false positives, and false negatives. For instance, the Autoencoder-LSTM model produced minimal false negatives, reinforcing its utility as a high-precision model that could reduce the risk of missed vulnerabilities. In contrast, the CodeBERT-CNN-BiLSTM model, while capturing nearly all vulnerable contracts, demonstrated a tendency for higher false positives, suggesting its application as an initial screening layer to maximise recall, followed by a refinement layer to filter false alerts.

The ROC-AUC scores, illustrated in the ROC curves of Fig. 4 for each model, provide a visual assessment of class separability. The Autoencoder-LSTM and CodeBERT-Transformer Encoder models achieved high ROC-AUC values, reflecting their ability to distinguish between vulnerable and non-vulnerable contracts across varying decision thresholds. This high separability indicates that these models are less likely to misclassify contracts in real-world applications where such errors could lead to significant security risks.

These results reveal that each model offers distinct advantages depending on the specific requirements of a reentrancy detection pipeline. The CodeBERT-CNN-BiLSTM, with its high recall, could



(a) CNN_BiGRU with Attention Model

(b) Autoencoder_LSTM Model

(c) CodeBERT_CNN_BiLSTM Model

(d) CodeBERT_Transformer Encoder Model

Figure 3: Confusion matrix of four different hybrid models.

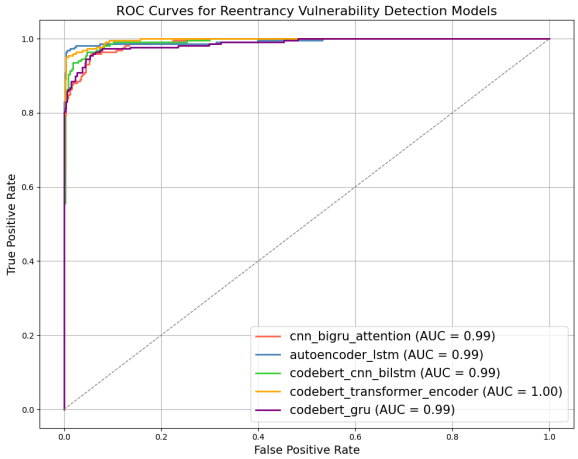


Figure 4: Receiver Operating Characteristic (ROC) curve for the ML models.

serve as a preliminary detection layer to ensure that potentially vulnerable contracts are flagged for further scrutiny. Subsequently, the Autoencoder-LSTM or CodeBERT-Transformer Encoder models, with their higher precision, could refine these initial predictions, minimising false positives and thereby enhancing the accuracy of the final output. This layered approach to vulnerability detection would allow the integration of multiple models, optimising detection while managing computational resources effectively.

5 CONCLUSION

This paper presents a comparative analysis of hybrid deep learning models for detecting reentrancy vulnerabilities in Ethereum smart contracts, employing architectures that leverage both semantic and structural information within the code. Among the models evaluated, the Autoencoder-LSTM and CodeBERT-Transformer Encoder demonstrated the highest overall performance, excelling in both accuracy and F1-score. These findings suggest that deep learning models integrating both sequence processing (e.g., LSTM, GRU) and contextual embeddings (e.g., CodeBERT) are particularly well-suited for the nuanced task of smart contract vulnerability detection, where both code structure and context-specific patterns play

a critical role. The implications of this study are significant for advancing smart contract security. By providing a reliable method for detecting reentrancy vulnerabilities, these models can be effectively integrated into blockchain development pipelines to proactively identify and mitigate potential security threats.

Our future work will focus on expanding the scope of this work beyond the specific subset of vulnerabilities and Ethereum contracts examined, which limits the generalisability of findings to other blockchain platforms and smart contract programming languages. The inclusion of a variety of dataset and vulnerabilities across different blockchain environments would enhance the efficiency and applicability of the models. Likewise, exploring different scalability options could address the computational demands inherent in training DL models for blockchain security, making these approaches more accessible in resource-constrained settings, such as internet of things devices. This broader approach could support a more comprehensive and adaptive framework for smart contract vulnerability detection, ultimately advancing security practices across diverse blockchain ecosystems.

REFERENCES

[1] Shikah J Alsunaidi, Hamoud Aljamaan, and Mohammad Hammoudeh. 2024. MultiTagging: A Vulnerable Smart Contract Labeling and Evaluation Framework. *Electronics* 13, 23 (2024), 4616.

[2] Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. 2022. Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts. *Blockchain: Research and Applications* (2022).

[3] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014), 2–1.

[4] Jaeseung Choi, Doyeon Kim, Soomin Kim, Gustavo Grieco, Alex Groce, and Sang Kil Cha. 2021. SMARTIAN: Enhancing Smart Contract Fuzzing with Static and Dynamic Data-Flow Analyses. In *36th IEEE/ACM Intl. Conf. Automated Software Engineering (ASE)*. 227–239.

[5] Li Duan, Liu Yang, Chunhong Liu, Wei Ni, and Wei Wang. 2023. A New Smart Contract Anomaly Detection Method by Fusing Opcode and Source Code Features for Blockchain Services. *IEEE Trans. Network and Service Management* 20, 4 (2023), 4354–4368.

[6] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd Intl. conference on software engineering*. 530–541.

[7] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[8] KeXin Gong, Xiangmei Song, Na Wang, Chunyang Wang, and Huijuan Zhu. 2023. SCGformer: Smart contract vulnerability detection based on control flow graph and transformer. *IET Blockchain* 3, 4 (2023), 213–221.

[9] Google. 2024. Welcome to Colaboratory. Available at https://colab.research.google.com/, Accessed: 2024-10-16.

[10] Liuqing Han. 2024. Smart Contract Reentrancy Vulnerability Detection Based on CNN and LSTM-Attention. In *5th Intl. Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*. IEEE, 147–151.

[11] Tengyun Jiao, Zhiyu Xu, Minfeng Qi, Sheng Wen, Yang Xiang, and Gary Nan. 2024. A Survey of Ethereum Smart Contract Security: Attacks and Detection. *Distrib. Ledger Technol.* 3, 3, Article 23 (Sept. 2024), 28 pages. https://doi.org/10.1145/3643895

[12] Olamide Jogunola, Bamidele Adebisi, Thokozani Shongwe, and Akilu Yunusa-Kaltungo. 2024. Security of Blockchain-Based Applications: A Case of Distributed Energy Systems. In *Key Themes in Energy Management: A Compilation of Current Practices, Research Advances, and Future Opportunities*. Springer, 397–414.

[13] Ningran Li, Minfeng Qi, Zhiyu Xu, Xiaogang Zhu, Wei Zhou, Sheng Wen, and Yang Xiang. 2025. Blockchain Cross-Chain Bridge Security: Challenges, Solutions, and Future Outlook. *Distrib. Ledger Technol.* 4, 1, Article 8 (Feb. 2025), 34 pages. https://doi.org/10.1145/3696429

[14] Jiayu Liang and Yuqing Zhai. 2023. SCGRU: A Model for Ethereum Smart Contract Vulnerability Detection Combining CNN and BiGRU-Attention. In *8th Intl. Conf. Signal and Image Processing (ICSIP)*. IEEE, 831–837.

[15] Jian-Wei Liao, Tsung-Ta Tsai, Chia-Kang He, and Chin-Wei Tien. 2019. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In *Sixth Intl. Conf. Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE, 458–465.

[16] Ruijie Luo, Feng Luo, Bingsen Wang, and Ting Chen. 2022. Smart contract vulnerability detection based on variant LSTM. In *Proceedings of the 2022 Intl. Conf. Big Data, IoT, and Cloud Computing*. 1–4.

[17] Mohamed Mahmoud, Mahmoud Kasem, Abdelrahman Abdallah, and Hyun Soo Kang. 2022. Ae-lstm: Autoencoder with lstm-based intrusion detection in iot. In *Intl. Telecomms. Conf. (ITC-Egypt)*. IEEE, 1–6.

[18] Chibuzo Obi-Okoli, Olamide Jogunola, Bamidele Adebisi, and Mohammad Hammoudeh. 2023. Machine Learning Algorithms to Detect Illicit Accounts on Ethereum Blockchain. In *Proceedings of the 7th Intl. Conf. Future Networks and Distributed Systems*. 747–752.

[19] Peng Qian, Zhenguang Liu, Yifang Yin, and Qinming He. 2023. Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode. In *Proceedings of the ACM Web Conference 2023*. 2220–2229.

[20] Elnaz Rabieinejad, Abbas Yazdinejad, Reza M. Parizi, and Ali Dehghantanha. 2023. Generative Adversarial Networks for Cyber Threat Hunting in Ethereum Blockchain. *Distrib. Ledger Technol.* 2, 2, Article 9 (June 2023), 19 pages. https://doi.org/10.1145/3584666

[21] Youcheng Shan. 2023. Social Network Text Sentiment Analysis Method Based on CNN-BiGRU in Big Data Environment. *Mobile Information Systems* (2023).

[22] Danish Vasan, Ebtesam Jubran S Alqahtani, Mohammad Hammoudeh, and Adel F Ahmed. 2024. An AutoML-based security defender for industrial control systems. *International Journal of Critical Infrastructure Protection* 47 (2024), 100718.

[23] Meiying Wang, Zheyu Xie, Xuefan Wen, Jianmin Li, and Kuanjiu Zhou. 2023. Ethereum smart contract vulnerability detection model based on triplet loss and BiLSTM. *Electronics* 12, 10 (2023), 2327.

[24] Zhiqiang Wang, Qingyun She, PengTao Zhang, and Junlin Zhang. 2020. Correct normalization matters: Understanding the effect of normalization on deep neural network models for click-through rate prediction. *arXiv preprint arXiv:2006.12753* (2020).

[25] Guangxia Xu, Lei Liu, and Zhaojian Zhou. 2022. Reentrancy vulnerability detection of smart contract based on bidirectional sequential neural network with hierarchical attention mechanism. In *2022 Intl. Conf. Blockchain Technology and Information Security (ICBCTIS)*. IEEE, 56–59.

[26] Lejun Zhang, Weijie Chen, Weizheng Wang, Zilong Jin, Chunhui Zhao, Zhennao Cai, and Huiling Chen. 2022. Cbgru: A detection method of smart contract vulnerability based on a hybrid model. *Sensors* 22, 9 (2022), 3577.

[27] Zibin Zheng, Neng Zhang, Jianzhong Su, Zhijie Zhong, Mingxi Ye, and Jiachi Chen. 2023. Turn the rudder: A beacon of reentrancy detection for smart contracts on ethereum. In *IEEE/ACM 45th Intl. Conf. Software Engineering (ICSE)*. IEEE, 295–306.

[28] Huijuan Zhu, Kaixuan Yang, Liangmin Wang, Zhicheng Xu, and Victor S Sheng. 2023. GraBit: A Sequential Model-Based Framework for Smart Contract Vulnerability Detection. In *EEE 34th Intl. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 568–577.