https://dx.doi.org/10.4314/dujopas.v10i3b.11
ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 10 No. 3b September 2024*

CodeELECTRA: An ELECTRA-based Approach for Improved Vulnerability Detection in Blockchain Smart Contracts

***Usman Bukar Usman [1], Kabir Umar[2], Aliyu Isah Agaie[3]**

[1]Department of Computer Science,
Faculty of Computing,
Bayero University Kano,
Nigeria.

[2]Department of Software Engineering,
Faculty of Computing,
Bayero University Kano,
Nigeria.

[3]Department of Information and Media Studies,
Faculty of Communication,
Bayero University Kano,
Nigeria.

Email: ubu1700016.msc@buk.edu.ng

Abstract

Blockchain technology has gained significant traction due to its core features of immutability, transparency, and decentralization. Smart contracts, self-executing programs stored on blockchains, play a vital role in enabling secure and automated transactions. Secure and automated transactions are made possible by self-executing programs and smart contracts that are kept on blockchains. The rapid progress of blockchain technology has been linked to an increase in security concerns targeting smart contracts. In comparison to traditional approaches, deep learning and transformer-based approaches have recently demonstrated a number of advantages, such as the capacity to learn from enormous datasets of known vulnerabilities and adjust to novel attack patterns. But Masked token training is the source of inefficiency for transformer-based approaches like CodeBert, resulting in low accuracy and restricted vulnerability coverage. Furthermore, we propose a novel approach, CodeELECTRA, by utilizing the Electra approach and context-aware masking to discover vulnerabilities, The model first step involves compiling and labeling the dataset of Solidity code that is vulnerable, and this is known as preprocessed Solidity code. Next, the logic decides which tokens to mask, the contest-aware masking step which employs a technique known as context-aware masking to strategically *mask* specific *portions of the code during training. In the third step, model will use the pre-trained ELECTRA model to learn contextual representations of the masked code. The masked code is fed into the ELECTRA encoder to generate contextual embedding, and the fully connected layer is employed in the final step to compare and adjust the ELECTRA models' output in order to classify vulnerabilities. The effectiveness of the chosen model in identifying vulnerabilities will evaluate using the Sodifi benchmark dataset. CodeELECTRA approach will improve vulnerability detection in blockchain smart contracts.*

Keywords: Blockchain technology, Smart contracts, Deep learning, Transformer, Vulnerability detection.

Author for Correspondence

INTRODUCTION

Blockchain has become an integral part of modern day-to-day lives. Blockchain technology (BCT) has revolutionized new ways to create and exchange digital assets, including electronic documents and cryptocurrency such as Bitcoin and Ether (Kongmanee et al., 2019). Blockchain systems can be defined as a distributed ledger implementation where transaction records are kept and connected via cryptography. Peers can join this peer-to-peer network and acquire a copy of the blockchain's current state, which supports these kinds of systems. A consensus protocol is utilized by the system peers to decide whether to accept or reject a new transaction based on a consensus (Vidal et al., 2023). BCT completely changed the sector and brought about a significant shift in the commercial world. By delivering trust, security, and transparency, it introduces new breakthroughs in banking, agriculture, healthcare, supply chains, and other fields (Pham Trong Linh & Minh Thanh, 2023).

On the most famous blockchain platform, Ethereum, there are already up to 1.5 million smart contracts running on the application (Cai et al., 2023). A smart contract's source code is frequently converted into bytecode before being implemented on the blockchain. Blockchain smart contracts are automated, decentralized applications on the blockchain that describe the terms of the agreement between buyers and sellers, reducing the need for trusted intermediaries and arbitration. Without a centralized authority, legal framework, or outside enforcement mechanism, smart contracts enable trusted transactions and agreements to be carried out between dispersed, anonymous individuals. Smart contracts work by following simple "if/when…then…" statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket (Kongmanee et al., 2019).

Vulnerability detection aims to improve the security of smart contracts. This can be achieved by looking into potential weaknesses and comparing the smart contracts to a list of previously established and well-known vulnerability patterns. (Sun et al., 2023) Currently, for the detection of smart contract vulnerabilities, many vulnerability detection tools have been proposed that combine more maturely developed detection methods. From the perspective of technology-driven development, smart contract vulnerability detection can be divided into traditional methods and machine learning, where traditional methods are driven by expert experience in development. In contrast, machine learning methods are driven by data (Chu et al., 2023). The specific techniques in this category are classical machine learning models, deep learning models, and ensemble learning models.

The branch of computer science known as "machine learning" focuses on teaching computers how to learn independently from data via experience (Zhang & Liu, 2022). Generally, machine learning methods retrieve the associated smart contract attributes and subsequently employ the machine learning algorithm's training classification model to identify vulnerabilities. For example, Xie et al. (2023) proposed Block-gram: mining knowledgeable features for efficient smart contract vulnerability detection. The majority of security vulnerability detection techniques use supervised learning, which finds vulnerabilities by training known vulnerability classifiers. The feature extraction methods of these methods are mostly based on CFG (Control Flow Graph) or AST (Abstract Syntax Tree), focusing on the analysis of smart contract operation code, source code, or byte code. Text mining (Sun et al., 2023). The two primary features utilized for smart contract vulnerability identification are text mining features and smart contract security metrics. ML-based methods offer several advantages over traditional techniques, including the ability to learn from large datasets of known vulnerabilities, adapt to new attack patterns, and handle complex and unstructured data.

(Vidal et al., 2023) In this context, most techniques are based on supervised learning, which involves using labeled datasets to train algorithms that classify data or predict outcomes for a particular output.

The use of deep learning techniques for smart contract vulnerability detection has gained popularity in recent years due to the field's rapid development. The extraction of features and the volume of data utilized for training are critical to the efficacy of vulnerability detection for smart contracts, according to the data-driven deep learning technique (Deng et al., 2023). For example, Liu et al. (2023) proposed Vulnerable Smart Contract Function Locating Based on a Multi-Relational Nested Graph Convolutional Network. While Rossini et al. (2023) proposed the use of deep neural networks for security vulnerability detection in smart contracts, Smart contract security based on deep learning vulnerability detection solutions can solve the issues of limited automation, low efficiency, and reliance on specialized knowledge of conventional detection techniques. However, most deep learning-based methods currently in use consider the limitations of the available data, struggle to mine information about smart contract vulnerabilities, have limited scalability, and ignore multimodal features. Transformers are a type of deep learning architecture that are particularly good at jobs involving natural language processing (NLP). They are able to spot patterns and examine relationships within sequences. The model analyzes the code, extracting its semantic properties and detecting vulnerabilities. Transformer models are influencing smart contract vulnerability detection in a big way (Ren et al., 2023). Transformers are excellent at figuring out intricate coding patterns, which makes them useful for deciphering smart contracts and spotting security flaws. For example, Tang et al. (2023) proposed a deep learning-based solution for smart contract vulnerability detection, while Sun et al. (2023) proposed ASSBert: an active and semi-supervised bert for smart contract vulnerability detection, and Jeon et al. (2023) proposed SmartConDetect: a highly accurate smart contract code vulnerability detection mechanism using BERT. Transformer models can be trained on smart contract bytecode and source code. This combined method, known as multimodal learning, offers a more comprehensive picture for vulnerability discovery (Chu et al., 2023).

The vulnerability problem with blockchain smart contracts drawn the attention of more academics due to the frequency of blockchain security incidents and the growing amount of assets engaged in smart contracts (Deng et al., 2023). A rising number of security risks aimed at smart contracts have been associated with the rapid advancement of blockchain technology. Malicious actors may use these vulnerabilities to hack into systems, steal currency, or alter data. Traditional methods for finding vulnerabilities in smart contracts, including fuzzy testing, dynamic analysis, static analysis, and symbolic execution, are still widely used to detect vulnerabilities in smart contracts. Although the traditional methods can be computationally expensive and have difficulties with complex contracts, they may not be effective for all types of vulnerabilities, particularly logic-based ones (Guidi & Michienzi, 2023). Manual code review is time-consuming and vulnerable to errors; it typically has low automation, low efficiency, and long detection times because it relies on expert experience. The machine learning approach offered several advantages over traditional approach, including the ability to learn from large datasets of known vulnerabilities, adapt to new attack patterns, and handle complex and unstructured data. Tang et al. (2023) proposed their approach, Lightning Cat, a deep learning-based approach for smart contract vulnerability detection. Although the above-mentioned existing work addressed the problem of vulnerability through deep learning, However, they have a notable weakness that demands attention. The approach specifically suffers from inefficiency due to model-masked token training, which leads to limited vulnerability coverage and low accuracy. Additionally, the models might not be specifically designed for Solidity code. A clear plan must be in place for choosing which code entities to mask while maintaining the syntactical validity of the masked code. potentially leading to missed vulnerabilities.

This paper proposes a deep learning solution with a model with a more sample-efficient pre-training task called CodeELECTRA. Based on the ELECTRA algorithm (Clark et al., 2020), the algorithm is more efficient than other pre-training tasks (Tepecik & Demir, 2024). which are trained to detect vulnerabilities in smart contract. The proposed CodeELECTRA pre-training model. was employed to preprocess the data, enhancing the semantic understanding and analysis capabilities of the Solidity code. By leveraging ELECTRA algorithm with context-aware masking and identify vulnerabilities through the CodeELECTRA model.

PROPOSED MODEL

In this section, we will fine-tune the deep learning algorithm Electra model with context-aware masking techniques to carry out the smart contract vulnerability detection task. In particular, we proposed a new CodeELECTRA model that captures the semantic features of smart contract vulnerabilities through effective fine-tuning strategy for Electra model, which solves the inconsistency and context-aware masking. Our proposed model is divided into five stages. In section 3.1, we will first give a summary of our framework solution. The next parts will provide a thorough explanation of the procedure.

Overview

Here, we've gone through multiple steps with our vulnerability detection mechanism in the smart contract process. In order to obtain the contract fragment, we must first preprocess the original smart contract source code file by doing the following steps:

Data cleaning is required in order to: (a) remove blank lines and unnecessary comments; (b) remove redundant information that does not affect the state of the contract; and (c) format the cleaned contract code to make it easier to use.

Tokenization: Tokens are the fundamental building blocks of the Solidity code that ELECTRA will understand, as shown in Figure 1. To do this, the code must be divided into function calls, operators, keywords, and identifiers. Etc.

Encoding: Run the preprocessed code sequence through the ELECTRA model, including any tokens with special tokens, and maybe tokens that are masked. After that, the model will produce an encoding, or vector representation, for the complete code sample. This encoding takes into account the context given by the surrounding tokens (even when they are hidden) to capture the semantic meaning and links between various code segments.

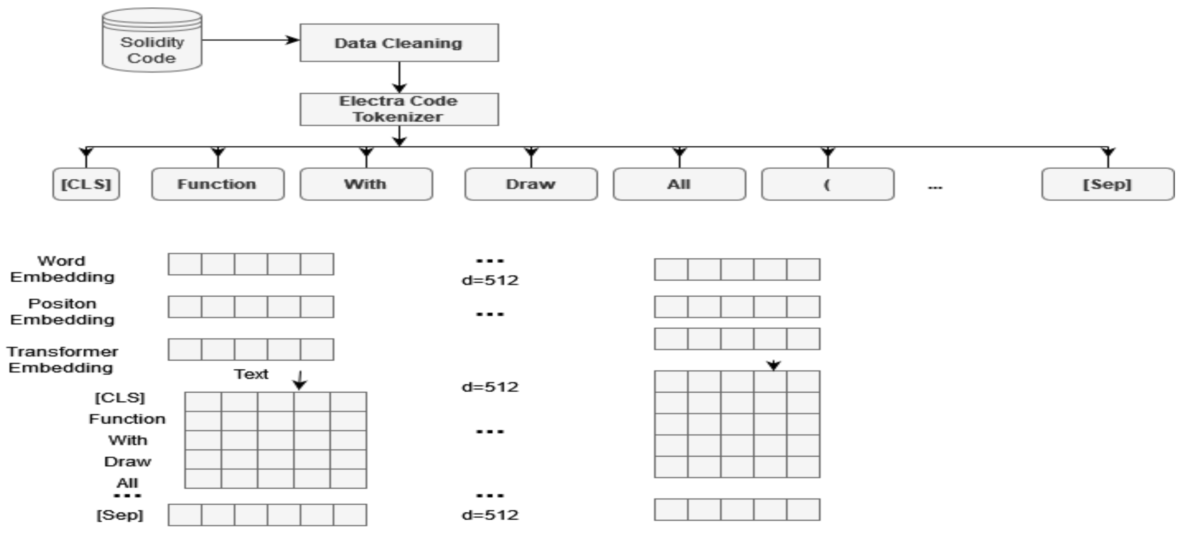

Figure 1: Model Tokenization (Tang et al., 2023)

CodeELECTRA MODEL DEVELOPMENT

The proposed model in figure 2. is divided into four stages. Preprocessed Solidity code, collecting and preparing the labeled dataset of Solidity code that is vulnerable, includes the first step. Contest-aware masking in the second stage; the third stage is the Electra model; and the final stage is the fully connected layer for vulnerability classification by fine-tuning the ELECTRA models and comparing their results. The Sodifi-benchmark dataset is used to evaluate the chosen model's efficacy in identifying vulnerabilities.

Preprocessed source code

Preprocessing the source code is necessary to remove sections that are unrelated to the contract execution logic and do not alter the smart contract's state, while also retaining the statements that are most closely linked to the vulnerability. pre-process and clean the data. This may include cleaning up irrelevant information, standardizing the formatting of the code, and fixing any inconsistencies. We will enumerate the following elements that must be removed from the source file, contract level, and function level in accordance with the development document for Ethereum's official programming language solidity.

Context-aware masking layer:

The model will analyze the preprocessed code using a chosen technique attention mechanism to masks out specific sections of the code based on the analysis. This can entail determining whether particular code elements like comments or function calls need to be hidden and creating context-aware masking techniques that ensure syntactic correctness. Based on the token classifications and surrounding context, the logic determines which tokens to mask. CodeELECTRA uses a method called context-aware masking to strategically conceal specific portions of the code during training. This enhances the model's capacity to generalize to unobserved code and helps it concentrate on critical components for vulnerability detection. Token Classification attention mechanism: The attention mechanism within the Electra model analyzes the code and classifies each token into a predefined category (keyword, function call, variable, etc.). Mask Tokens: Specific tokens or parts of tokens are masked in the original code sequence, frequently substituted with a particular ID [MASK]. Masked Code Sequence: The resulting sequence depicts the code with masked elements, focusing on crucial places for vulnerability identification. The attention scores from the self-attention mechanism are used to determine which tokens to mask. Tokens with higher attention scores from relevant context (e.g., function call arguments) are more likely to be masked. Tokens with lower scores (e.g., function names, control flow keywords) are less likely to be masked. The approach dynamically adapts the masking based on the code's specific context, potentially leading to a more nuanced understanding of vulnerabilities.

Equations in Context-Aware Masking:
Let the original code sequence be represented as
X = [x1, x2,..., xn] **eqn 1**
Where xi is a token. Using a masking predictor network, the model forecasts a masking probability score m_i for every token xi. This network considers the code sequence's present context. Based on the expected score m i, a masking function (often placing a threshold on the probability) decides whether a token is masked at all. The masked version of the code sequence (Y) can be expressed mathematically as follows:
Y = [y1, y2,..., yn] **eqn 2**
where yi = xi if m_i < threshold (not masked)
yi = [MASK] if m_i >= threshold (masked)

Electra model (fine tuning)
Pre-trained ELECTRA model for code analysis it will Takes the masked code as input. Processes the code through its encoder layers, which include: Self-Attention sub-layer: Identifies relationships between masked and unmasked parts of the code. Feed Forward sub-layer: Introduces non-linearity for complex relationship modeling. Generates a contextual embedding that captures the code's meaning despite masking. For fine-tuning we will Train the ELECTRA model on a dataset of labeled smart contracts with various vulnerability types. This dataset should encompass different categories of vulnerabilities we want the model to detect the following vulnerability, integer overflow, reentrancy, timestamp dependency, Unhandled exception, Tx.origin, unchecked sent and Transaction-ordering dependence (TOD). The Model will use the pre-trained ELECTRA model to learn contextual representations of the masked code. The masked code is fed into the ELECTRA encoder to generate contextual embedding.

Fully connected layers
In order to output the classification labels, we will add fully connected layers. First, we added a new linear layer with features on top of the existing linear layer. We then utilized the ReLU activation function to avoid the limited capacity of a single linear layer. Next, we will introduce a dropout layer with a 0.1 dropout rate after the activation layer to prevent overfitting. Finally, we used a linear layer with four features for the output. The parameters of these new layers were updated during the fine-tuning process. Attention weights, represented as wij, are determined by the multi-head self-attention mechanism for every point i in the input code sequence. The following formula is used to calculate the attention weights:

$$w_{ij} = \textbf{Softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d}}\right) \quad \textbf{eqn 3}$$

In this case, d is the dimension of the queries and keys, qi is the query at position I, and kj is the key at position J. By multiplying the attention weights wij by the corresponding values vj and adding them up, the output of the self-attention mechanism at location i, represented as oi, is obtained:

$$Oi = \sum_{j=1} w_{ij} \cdot vj1e \quad \textit{eqn 4}$$

A feed-forward neural network sub-layer is also present in each encoder layer. It uses the following equation to process the output of the self-attention sub-layer:

$$\text{FFN}(x) = \text{ReLU}(x \cdot W_1 + b_1) \cdot W_2 + b_2 \quad \textbf{eqn 5}$$

Here, W_1,b_1, and W_2 ,b_2 are the feed-forward neural network's parameters, and x is the self-attention sub-layer's output.

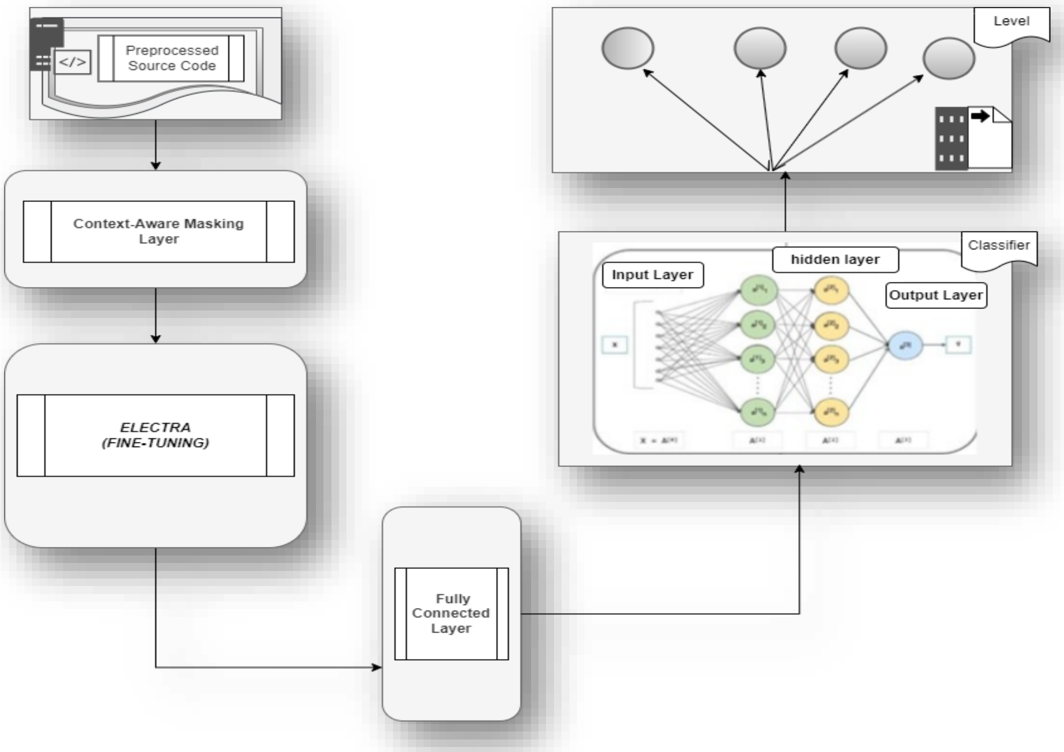

Figure 2: Our Proposed CodeELECTRA

PROPOSED EXPERIMENT

We will train and evaluate the methods in the same contexts to guarantee an equitable assessment of the various approaches. All tests will be conducted using the PyCharm program, the PyTorch framework, and the Python programming language on a Windows computer. The efficacy of CodeELECTRA for enhanced vulnerability detection in blockchain smart contracts will be demonstrated by this experiment. CodeELECTRA will have the potential to be a useful tool for improving blockchain application security by utilizing Electra's architecture and context-aware masking.

Parameter Settings

The key parameters to consider for our proposed CodeELECTRA experiment are in the Electra Base Model, Context-Aware Masking and fully connected layer.

i. Electra Base Model

The number of encoder layers determines how well the model can understand intricate relationships found in code. We try to vary the values (e.g., 12 or 24) according to the size of our dataset and available computing power.

Hidden size: The model will learn the dimensionality of the hidden representations. While higher values can enhance learning, they will also lengthen the training period. A possible typical range is likely to remain 128–768."

Dropout rate: During training, neurons will be randomly dropped out to help prevent overfitting. Standard values can be adjusted based on validation performance and will be initialized at around 0.1.

ii. Context-Aware Masking

Learning-Based Masking Probability Threshold: This threshold establishes the point at which a predicted masking probability is sufficiently high to effectively mask the token. We Try varying the values (e.g., 0.5, 0.8) until you find the best balance between hiding significant and irrelevant information.

iii. Fully connected layer

Learning rate will be scheduled to update the model weights during training. A tiny value of 1e-5 will be used initially and will adapt according to validation performance and convergence pace. An Optimizer Will Be Selected to aid in the model's effective convergence, such as Adam or RMSprop. A Loss Function will be employed that calculates the prediction error of the model. For binary categorization (vulnerable vs. non-vulnerable), binary cross-entropy will likely be utilized.

PERFORMANCE METRICS

Several performance metrics, such as accuracy, F1 score, recall, and precision, will be utilized for evaluating the methods (Sun et al., 2023). In actuality, accuracy will be defined as the proportion of accurately predicted cases (including true positives and true negatives) to the total number of cases. It will offer an overall general measure of accuracy. The matrix will contrast the predicted value of the machine learning model with the actual goal value.

The accuracy of a classifier will simply be the frequency with which it makes the correct prediction, as shown in eqn. 6.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}}$$

Eqn. 6

Precision, which is often referred to as positive predictive value, quantifies the percentage of accurately predicted positive instances, or true positives, among all instances that are projected to be positive. The accuracy of optimistic predictions is the main focus. Is described in Eqn. 7 below.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Eqn. 7

Recall computes the percentage of accurately anticipated positive cases (true positives) out of all actual positive instances. It is sometimes referred to as sensitivity or a true positive rate. It centers on the model's capacity to recognize every positive example. Is defined in Eqn. 8

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Eqn. 8

F1 score, another significant metric that combines recall and precision is the **F1 score**. It offers a balance between the two while taking into account their trade-offs. When both precision and recall are crucial, or when the dataset is unbalanced, the F1 score is especially helpful. Is described in Eqn. 9.

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Eqn. 9

2.1 Dataset

Three most-used and public datasets were gathered in order to provide sufficient experimental data: Smartbugs (Durieux et al., 2020) . SoliAudit-benchmark, SoliAudit vulnerability analyzer dataset, and SolidiFi-benchmark. We will select seven categories of vulnerabilities—timestamp dependency, unhandled exception, transaction-ordering dependence (TOD), unchecked sent, integer overflow, reentrancy, and tx.origin, from the Solidify benchmark dataset.

EXPECTED RESULTS

CodeELECTRA is expected to identify vulnerabilities more accurately than baseline techniques, with metrics like recall, F1 score, accuracy, and precision. It is expected to work well on untested code with various vulnerabilities. Due to context-aware masking, allowing greater generalizability to real-world scenarios. Table 1. will show the presentation of the metric result data based on our evaluation.

Table 1. Expected Evaluation of the Metrics Results for Each Type of Vulnerability

Vulnerability	Method	Accuracy (%)	Precision (%)	Recall (%)	F1(%)
Reentrancy	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
Timestamp-Dependency	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
TOD	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
Unchecked sent	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
Integer overflew	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
Unhandled Exceptions	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X
Tx.origin	Optimized-CodeBERT	X	X	X	X
	Electra	X	X	X	X
	CodeElectra	X	X	X	X

CONCLUSION

The increasing adoption of blockchain technology and the rising value of assets stored on smart contracts highlight the critical need for robust security solutions. The security of blockchain systems could be greatly improved by leveraging their competence to learn from code with context-aware masking and identify vulnerabilities through CodeELECTRA. Once pre-trained, ELECTRA is fine-tuned on a dataset of labeled Solidity code with vulnerabilities. This fine-tuning helps the model focus on the specific features and patterns relevant to vulnerability detection. CodeELECRA will offer a viable method for enhancing blockchain smart contract vulnerability detection. This can foster trust and wider adoption of blockchain technology across various industries. The research findings can also will contribute to the development of more robust and secure smart contract development practices.

CONFLICTS OF INTEREST

No conflict of interest was declared by the authors

REFERENCES

Cai, J., Li, B., Zhang, J., Sun, X., Chen, B., 2023. Combine sliced joint graph with graph neural networks for smart contract vulnerability detection. Journal of Systems and Software 195, 111550. https://doi.org/10.1016/j.jss.2022.111550

Chu, H., 2023. A survey on smart contract vulnerabilities: Data sources, detection and repair. Information and Software Technology.

Clark, K., Luong, M.T., Le, Q.V. and Manning, C.D., 2020. Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555. Deng, W., Wei, H., Huang, T., Cao, C., Peng, Y., Hu, X., 2023. Smart Contract Vulnerability Detection Based on Deep Learning and Multimodal Decision Fusion.

Durieux, T., Ferreira, J.F., Abreu, R. and Cruz, P., 2020, June. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In Proceedings of the ACM/IEEE 42nd International conference on software engineering (pp. 530-541).

Guidi, B., Michienzi, A., 2023. Delving NFT vulnerabilities, a sleepminting prevention system. Multimed Tools Appl 82, 46065–46084. https://doi.org/10.1007/s11042-023-16087-1

Jeon, S., Lee, G., Kim, H. and Woo, S.S., 2021, August. Smartcondetect: Highly accurate smart contract code vulnerability detection mechanism using bert. In *KDD Workshop on Programming Language Processing*.

Kongmanee, J., Kijsanayothin, P., Hewett, R., 2019. Securing Smart Contracts in Blockchain, in: 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). Presented at the 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), IEEE, San Diego, CA, USA, pp. 69–76. https://doi.org/10.1109/ASEW.2019.00032

Liu, Z., Qian, P., Yang, J., Liu, L., Xu, X., He, Q., Zhang, X., 2023. Rethinking Smart Contract Fuzzing: Fuzzing With Invocation Ordering and Important Branch Revisiting.

Pham Trong Linh, Minh Thanh, T., 2023. Proposing of Imaging Graph Neural Network with Defined Security Pattern for Improving Smart Contract Vulnerability Detection. ICT Research 70–79. https://doi.org/10.32913/mic-ict-research.v2023.n2.1198

Ren, X., Wu, Y., Li, J., Hao, D., Alam, M., 2023. Smart contract vulnerability detection based on a semantic code structure and a self-designed neural network. Computers and Electrical Engineering 109, 108766. https://doi.org/10.1016/j.compeleceng.2023.108766

Rossini, M., Zichichi, M., Ferretti, S., 2023. On the Use of Deep Neural Networks for Security Vulnerabilities Detection in Smart Contracts, in: 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops). Presented at the 2023 IEEE International Conference on

Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), IEEE, Atlanta, GA, USA, pp. 74–79. https://doi.org/10.1109/PerComWorkshops56833.2023.10150302

Sun, X., Tu, L., Zhang, J., Cai, J., Li, B., Wang, Y., 2023. ASSBert: Active and semi-supervised bert for smart contract vulnerability detection. Journal of Information Security and Applications 73, 103423. https://doi.org/10.1016/j.jisa.2023.103423.

Smartbugs-wild. https://github.com/smartbugs/smartbugs-wild (2019).

Tang, X., Du, Y., Lai, A., Zhang, Z., Shi, L., 2023. Lightning Cat: A Deep Learning-based Solution for Smart Contracts Vulnerability Detection (preprint). In Review. https://doi.org/10.21203/rs.3.rs-3104649/v1

Tepecik, A., Demir, E., 2024. Emotion Detection with Pre-Trained Language Models BERT and ELECTRA Analysis of Turkish Data.

Vidal, F.R., Ivaki, N., Laranjeiro, N., 2023a. OpenSCV: An Open Hierarchical Taxonomy for Smart Contract Vulnerabilities.

Vidal, F.R., Ivaki, N., Laranjeiro, N., 2023b. Advancing Blockchain Security: from Vulnerability Detection to Transaction Revocation, in: 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S). Presented at the 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S), IEEE, Porto, Portugal, pp. 179–181. https://doi.org/10.1109/DSN-S58398.2023.00048

Xie, X., Wang, H., Jian, Z., Fang, Y., Wang, Z., Li, T., 2023. Block-gram: Mining knowledgeable features for efficiently smart contract vulnerability detection. Digital Communications and Networks S2352864823001347. https://doi.org/10.1016/j.dcan.2023.07.009

Zhang, Y., Liu, D., 2022. Toward Vulnerability Detection for Ethereum Smart Contracts Using Graph-Matching Network. Future Internet 14, 326. https://doi.org/10.3390/fi14110326