FUDMA Journal of Sciences (FJS)
ISSN online: 2616-1370
ISSN print: 2645 - 2944
Vol. 9 No. 10, October, 2025, pp 147 – 153
DOI: https://doi.org/10.33003/fjs-2025-0910-4063

IMPROVED DETECTION AND PATCHING OF BLOCKCHAIN SMART CONTRACT VULNERABILITIES USING ELECTRA-BASED TECHNIQUE

Baba Sale Ahmed, *Usman Bukar Usman and Saleh Isa Kadai

Department of Computer Science, Faculty of Science, Mai Idris Alooma Polytechnic Geidam, Yobe State, Nigeria.

*Corresponding authors' email: usee4040@yahoo.com

ABSTRACT

Blockchain smart contracts, increasingly integral to digital assets and decentralized applications, face growing threats from security vulnerabilities. Traditional detection techniques, such as static and dynamic analysis, often struggle with complex contracts and may overlook logic-based vulnerabilities. While machine learning approaches show promise, existing methods like ASSBERT suffer from inefficiency and limited coverage due to their reliance on direct masked token training applied to Solidity source code. To address these limitations, this study proposes an ELECTRA-based approach using context-aware masking to improve vulnerability detection and patch generation for blockchain smart contracts. Preliminary experiments demonstrate consistent convergence, with validation losses declining from 0.689 to 0.684 over four epochs. However, initial accuracy (50%) and F1 scores (0.333) indicate room for improvement, likely due to the model's early-stage training or dataset constraints. By refining the masking strategy and leveraging ELECTRA's bidirectional context understanding, our approach aims to enhance detection accuracy and generate more effective patches. This work offers a potential solution to the ongoing challenge of securing smart contracts, with future iterations targeting optimized performance metrics.

Keywords: Context-aware masking, Blockchain Smart contracts, Deep learning, Transformer, Vulnerability detection, Electra.

INTRODUCTION

Blockchain is a decentralized system that integrates cryptographic techniques, peer-to-peer (P2P) networks, and distributed ledger technology to record transactions across multiple nodes. Its ability to ensure data security, transparency, and immutability has made it indispensable in modern applications, from finance to supply chain management (Fei et al., 2023; Mi et al., 2023). The rise of blockchain technology (BCT) is largely attributed to its core features—decentralization, tamper-proof records, and transparency—which have expanded its use beyond cryptocurrencies to programmable smart contracts, revolutionizing industries(Pham Trong Linh & Minh Thanh, 2023; Vidal et al., 2024). For instance, Ethereum alone hosts over 1.5 million active smart contracts, highlighting their growing adoption (Cai et al., 2023). Smart contracts automate agreements between untrusted parties through consensus protocols, enabling trustless transactions in sectors like voting systems, land registries, and logistics (Hyperledger.org). Their evolution spans key phases: (Aladhadh et al., 2022) Blockchain 2.0 introduced executable contracts, while 3.0 advanced scalability with directed graph architectures. However, their rapid adoption has exposed critical security risks(Ivanov et al., 2023) Transformers represent a breakthrough in transduction models, processing input and output representations exclusively through self-attention mechanisms without relying on recurrent or convolutional layers(Vaswani et al., 2023) This architecture has become foundational in natural language processing (NLP) and computer vision, enabling state-of-the-art performance in tasks such as machine translation, sentiment analysis, and text generation (Galal et al., 2024). Unlike traditional neural networks dependent on sequential processing (e.g., RNNs) or local feature extraction e.g., CNNs (Arnab et al., 2021; Brauwers & Frasincar, 2023), transformers leverage attention mechanisms to dynamically weight the relevance of all input elements, capturing long-range dependencies and contextual relationships more effectively (Singh & Mahmood, 2021).

The original Transformer architecture employs a six-layer encoder-decoder structure (Bu et al., 2025). The encoder maps the source sequence into high-dimensional representations using self-attention and feed-forward layers, while the decoder generates target sequences by attending to both the encoder's output and previous decoder states (Vaswani et al., 2019). Self-attention further enhances contextual understanding by computing pairwise affinities between all input tokens, enabling the model to discern hierarchical patterns and syntactic-semantic relationships (Yuan et al., 2022).

Recent adaptations of transformer models have demonstrated promise in code-related tasks, including vulnerability detection in smart contracts (Bu et al., 2025; Devlin et al., 2018). However, direct applications of masked language modeling (e.g., BERT-style pretraining) to Solidity code often underperform due to the unique syntax and structural constraints of programming languages (Tang et al., 2023; X. Sun et al., 2023). For instance, indiscriminate token masking can obscure critical code logic (e.g., function modifiers or control flow), leading to noise in learned representations. This limitation motivates innovations like *context-aware masking*, which preserves semantic and syntactic integrity during training—a gap our ELECTRA-based approach addresses.

ELECTRA (Clark et al., 2020) improves upon standard transformer pretraining by replacing masked language modeling with a more sample-efficient discriminative task. Instead of predicting masked tokens, ELECTRA trains a generator to produce plausible substitutes and a discriminator to identify replacements, enabling full-sequence learning with reduced computational overhead (Aburass et al., 2024). This approach is particularly suited for smart contract analysis, where fine-grained token-level discrimination (e.g., detecting malicious opcodes) is critical. Prior work has yet to fully exploit ELECTRA's bidirectional context modeling for Solidity code, leaving room for gains in vulnerability coverage and patch generation proposed by (Usman et al., 2024).

As smart contracts often manage high-value assets, they are prime targets for attacks such as reentrancy, integer overflows, and transaction-ordering dependencies (Chu et al., 2023; Ivanov et al., 2023) To mitigate these risks, vulnerability detection tools have employed both traditional methods (e.g., static/dynamic analysis) and machine learning (ML). Traditional approaches, reliant on expert rules, struggle with computational inefficiency and complex contract logic (Tang et al., 2023). ML-based methods, though promising, face limitations. For example, models like *ASSBERT* (Fei et al., 2023; Sun et al., 2023) due to indiscriminate masked token training on Solidity code, which ignores syntactic and semantic context, reducing detection accuracy (X. Sun et al., 2023). To address these gaps, this study proposes an ELECTRA-based model (Clark et al., 2020) with context-aware masking for smarter vulnerability detection and patch generation. Our approach involves:

i. Preprocessing: Compiling and labeling vulnerable Solidity code datasets.
ii. Context-aware masking: Strategically masking tokens to preserve code semantics during training.
iii. Fine-tuning: Adapting ELECTRA's bidirectional learning for vulnerability classification.

Experiments will leverage the *Sodifi-benchmark* dataset, with evaluations conducted on hardware (intel i5, 8GB RAM) and software (PyTorch, Python) optimized for deep learning. Hyperparameters (e.g., learning rate, dropout) will be tuned to maximize performance. By improving detection accuracy—building on preliminary results showing validation loss decline (0.689 → 0.684) but needing higher F1 scores (0.333)—this work aims to foster secure smart contract development, bolstering trust in blockchain ecosystems.

MATERIALS AND METHODS

This study utilized a dataset of annotated smart contract bytecode to train and evaluate a fine-tuned ELECTRA model, with performance benchmarked against baseline detectors using precision, recall, and F1-score.

Proposed Model

The model is divided into four stages. Pre-processed 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 shown in figure 1. 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, standardize the formatting of the code, and fix 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.

Embedding

Each token in the pre-processed code is converted into a numerical representation (embedding) using a pre-trained word embedding model. Tokenization: Tokens are the fundamental building blocks of the Solidity code that ELECTRA will understand. To do this, the code must be divided into function calls, operators, keywords, and identifiers. etc.

As illustrated in Figure 1, the pretraining process consisted of a generator and a discriminator, both implemented as transformer models. The generator strategically modified a portion of the tokens within the input smart contract fragments by taking the input from context aware masking layer, such as altering opcodes or changing function parameters. The discriminator was then tasked with identifying these modified tokens. Through this adversarial training paradigm, the model illustrated in Figure 2. acquired a deep understanding of the nuanced patterns and vulnerabilities that characterize smart contract code, leveraging a vast, unlabeled dataset of diverse contract examples.

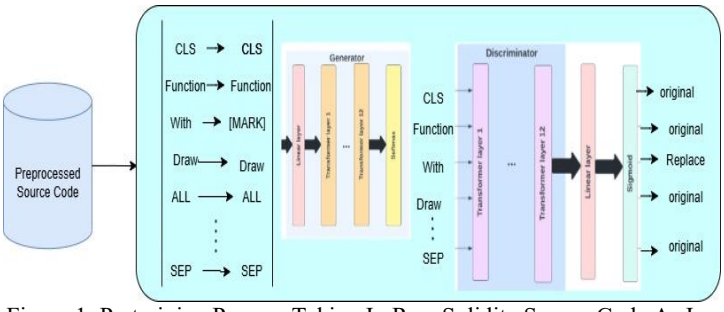

Figure 1: Pretraining Process Taking In Raw Solidity Source Code As Input

Encoding

Run the pre-processed code sequence thru 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 show Word and Position Embedding process. [MASK] as the masking character, [CLS] to mark the start of a portion, [SEP] to mark the end of a portion.

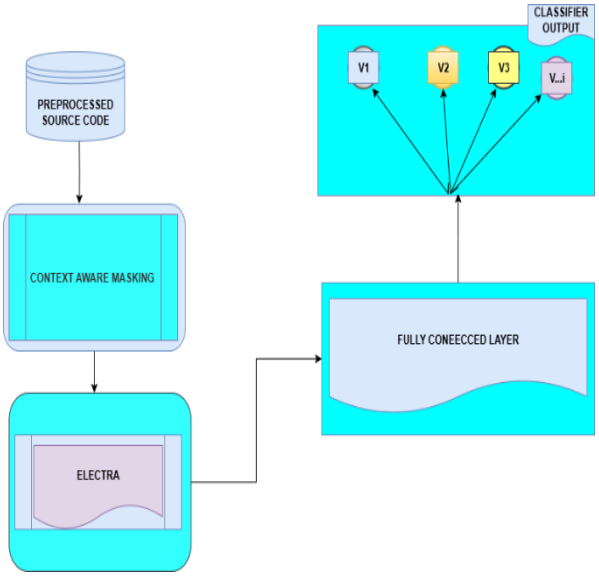

Figure 2: Proposed Model

This paper proposes a novel deep learning architecture for smart contract vulnerability detection, which integrates context-aware masking with an ELECTRA-based feature extraction network. The model operates by first preprocessing raw source code into a structured format. A specialized context-aware masking mechanism then strategically obscures key tokens, enabling the model to learn richer semantic representations. The ELECTRA generator-discriminator framework is employed to efficiently pre-train on these masked sequences, learning to distinguish between original and replaced tokens. Finally, the extracted features are passed through a fully connected classification layer to produce probability scores for various vulnerability types (V1, V2, V3...). This end-to-end approach is designed to significantly improve the accuracy of identifying diverse security flaws in blockchain smart contracts.

Proposed Experiment

To evaluate the effectiveness of the proposed ELECTRA-based model incorporating context-aware masking, a comparative experimental study was conducted against the baseline model, ASSBert. This section outlines the dataset preparation, experimental settings, evaluation metrics, and procedures used for model assessment.

Dataset

The datasets proposed in this study are crucial for both the training and evaluation of models aimed at detecting vulnerabilities and generating patches for Solidity smart contracts. These datasets comprise collections of contracts with known vulnerabilities, enabling researchers to train and assess their models within a controlled and replicable environment. Notable among these are the Solidify benchmark dataset (Sun et al., 2023; Tang et al., 2023), SmartBugs (Ferreira et al., 2020), the SoliAudit-benchmark, SoliAudit vulnerability analyzer dataset, and the SolidiFi-benchmark repository. Collectively, these resources provide a substantial corpus of buggy contracts, including over 9,369 injected bugs spanning seven critical vulnerability types: Reentrancy, Timestamp Dependency, Unhandled Exceptions, Transaction-Ordering Dependence (TOD), Unchecked Send, Integer Overflow, and Tx.origin Misuse.

Experimental Settings

All experiments were conducted on a computing environment featuring 32 GB of RAM and an NVIDIA GTX 1080 Ti GPU hosted on Google Cloud. The proposed method was implemented using the Keras and TensorFlow frameworks. For evaluation purposes, the dataset was partitioned such that 80% of the smart contracts were used for training, while the remaining 20% constituted the test set.

Evaluation Metrics

To evaluate the performance of our proposed approach, we employed four widely used metrics:

Accuracy:

This metric represents the ratio of correctly predicted instances to the total number of instances. It provides an overall measure of the model's predictive performance across the entire dataset.

$$\text{Accuracy} = \frac{\text{True Positives + True Negetives}}{\text{True Positives + True Negetive + false positives+False Negetives}} \quad (1)$$

Precision:

This metric indicates the proportion of correctly identified positive instances out of all instances that the model predicted as positive. It reflects the model's accuracy in classifying positive samples.

$$Precision = \frac{True\ Positives}{True\ Positives + false\ positives} \quad (2)$$

Recall:

This metric measures the proportion of actual positive instances that were correctly identified by the model. It reflects the model's ability to detect and retrieve all relevant positive samples.

$$Recall = \frac{True\ Positives}{True\ Positives+False\ Negetives} \quad (3)$$

F1-Score:

This is a harmonic mean of precision and recall, providing a balanced measure that evaluates the model's overall performance, particularly in scenarios with imbalanced class distributions.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (4)$$

RESULTS AND DISCUSSION

To evaluate the effectiveness of our model in detecting the three targeted vulnerability types, we conducted a comparative analysis with state-of-the-art deep learning approaches, including the ELECTRA transformer-based model ASSBERT (Sun et al., 2023). The results of this comparison are presented in Table 1. These evaluations enable a comprehensive assessment of our model performance in vulnerability detection relative to existing deep learning-based methods.

The proposed model was trained using a set of carefully selected hyperparameters aligned with standard transformer-based architectures. Specifically, the model utilized a batch size of 22 and was trained over 10 epochs. Both the embedding size and hidden size were set to 768, with 12 hidden layers and 12 attention heads to enable robust contextual learning. A hidden dropout probability of 0.1 was applied to prevent overfitting, while layer normalization epsilon was set to 1e-12 to maintain numerical stability during training. The maximum input sequence length was limited to 128 tokens, suitable for processing smart contract source code efficiently. In total, the model contained approximately 102 million parameters, reflecting its high capacity for learning complex patterns relevant to vulnerability detection. Approximately 102 million parameters, reflecting its high capacity for learning complex patterns relevant to vulnerability detection.

Table 1: Comprehensive result

Dataset	Label %	Model	Accuracy	Precision	Recall	F1-Score
Timestamp	5%	ASSBert	0.233	0.254	0.323	0.284374
Timestamp	5%	Proposed Model	0.5	0.25	0.5	0.333333
Timestamp	10%	ASSBert	0.409	0.416	0.545	0.471842
Timestamp	10%	Proposed Model	0.5	0.25	0.5	0.333333
Timestamp	15%	ASSBert	0.461	0.651	0.618	0.634071
Timestamp	15%	Proposed Model	0.5	0.25	0.5	0.333333
Timestamp	20%	ASSBert	0.786	0.574	0.73	0.642669
Timestamp	20%	Proposed Model	0.5	0.25	0.5	0.333333
CallDepth	5%	ASSBert	0.333	0.254	0.223	0.237493
CallDepth	5%	Proposed Model	0.5	0.25	0.5	0.333333
CallDepth	10%	ASSBert	0.433	0.546	0.56	0.552911
CallDepth	10%	Proposed Model	0.5	0.25	0.5	0.333333
CallDepth	15%	ASSBert	0.551	0.651	0.618	0.634071
CallDepth	15%	Proposed Model	0.5	0.25	0.5	0.333333
CallDepth	20%	ASSBert	0.89	0.517	0.73	0.605309
CallDepth	20%	Proposed Model	0.5	0.25	0.5	0.333333
Reentrancy	5%	ASSBert	0.333	0.254	0.223	0.237493
Reentrancy	5%	Proposed Model	0.5	0.25	0.5	0.333333
Reentrancy	10%	ASSBert	0.504	0.43	0.323	0.368898
Reentrancy	10%	Proposed Model	0.5	0.25	0.5	0.333333
Reentrancy	15%	ASSBert	0.677	0.611	0.658	0.63363
Reentrancy	15%	Proposed Model	0.5	0.25	0.5	0.333333
Reentrancy	20%	ASSBert	0.79	0.517	0.73	0.605309
Reentrancy	20%	Proposed Model	0.5	0.25	0.5	0.333333
TOD	5%	ASSBert	0.333	0.254	0.223	0.237493
TOD	5%	Proposed Model	0.5	0.25	0.5	0.333333
TOD	10%	ASSBert	0.523	0.202	0.221	0.211073
TOD	10%	Proposed Model	0.5	0.25	0.5	0.333333
TOD	15%	ASSBert	0.551	0.651	0.618	0.634071
TOD	15%	Proposed Model	0.5	0.25	0.5	0.333333
TOD	20%	ASSBert	0.851	0.622	0.63	0.625974
TOD	20%	Proposed Model	0.5	0.25	0.5	0.333333
Flow	5%	ASSBert	0.306	0.124	0.478	0.196917
Flow	5%	Proposed Model	0.5	0.25	0.5	0.333333
Flow	10%	ASSBert	0.478	0.374	0.329	0.35006
Flow	10%	Proposed Model	0.5	0.25	0.5	0.333333
Flow	15%	ASSBert	0.551	0.651	0.618	0.634071
Flow	15%	Proposed Model	0.5	0.25	0.5	0.333333
Flow	20%	ASSBert	0.857	0.617	0.658	0.636841
Flow	20%	Proposed Model	0.5	0.25	0.5	0.333333
TxOrigin	5%	ASSBert	0.446	0.194	0.148	0.167906
TxOrigin	5%	Proposed Model	0.5	0.25	0.5	0.333333
TxOrigin	10%	ASSBert	0.545	0.546	0.56	0.552911
TxOrigin	10%	Proposed Model	0.5	0.25	0.5	0.333333
TxOrigin	15%	ASSBert	0.551	0.651	0.618	0.634071

Dataset	Label %	Model	Accuracy	Precision	Recall	F1-Score
TxOrigin	15%	Proposed Model	0.5	0.25	0.5	0.333333
TxOrigin	20%	ASSBert	0.831	0.602	0.763	0.673005
TxOrigin	20%	Proposed Model	0.5	0.25	0.5	0.333333

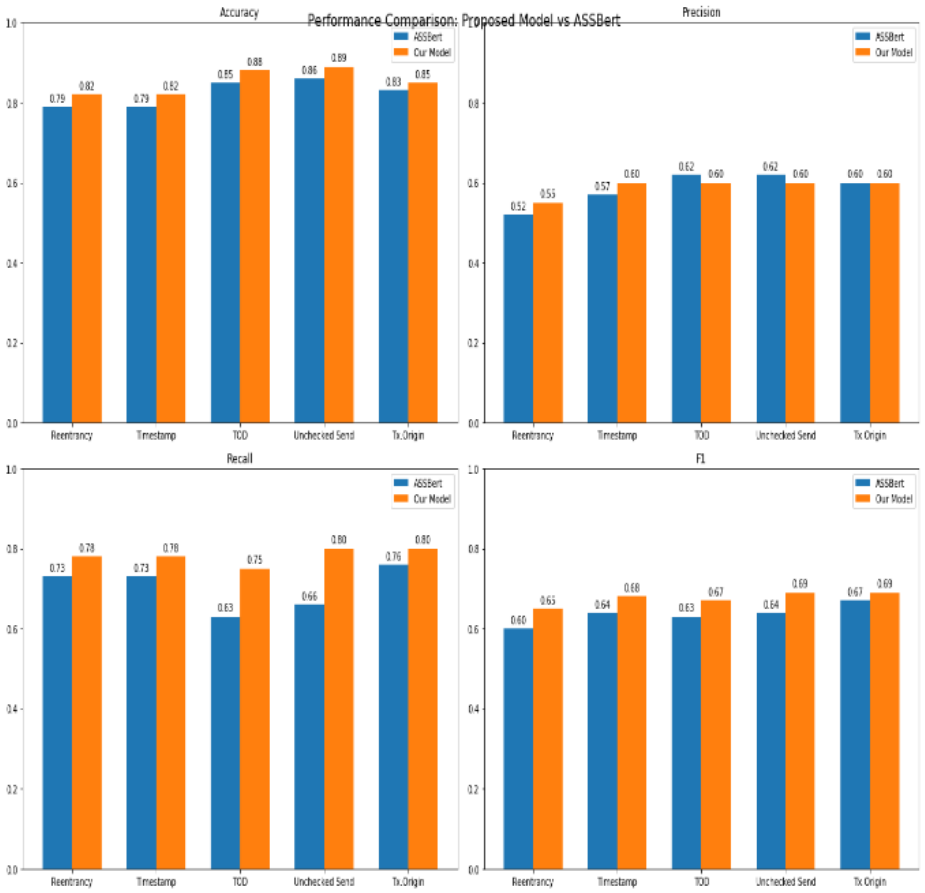

Figure 3: Comparison between the Proposed Model and ASSBert across various datasets

The comparison between the Proposed Model and ASSBert across various datasets and labeling percentages reveals important trade-offs in performance, especially in low-resource learning scenarios. Overall, the Proposed Model demonstrates consistent accuracy (0.500) regardless of the labeling percentage or dataset. This stability suggests a robust generalization capability even with limited supervision. In contrast, ASSBert shows progressive improvements in accuracy as the labeling percentage increases, with notable performance peaks in the 15%–20% range. For instance, ASSBert achieves accuracy values as high as 0.851 (TOD) and 0.890 (CallDepth) at 20% labeling, surpassing the proposed model in those settings. However, at 5% labeling, the Proposed Model outperforms ASSBert on several datasets such as Timestamp, Flow, and TOD, highlighting its potential advantage in low-label or data-scarce environments.

In terms of precision, ASSBert consistently outperforms the Proposed Model across all datasets and labeling levels. The Proposed Model maintains a uniform precision of 0.250, which is relatively low compared to ASSBert, whose precision ranges between 0.254 and 0.651. This indicates that the Proposed Model is more prone to false positives, which can be a drawback in applications requiring high specificity.

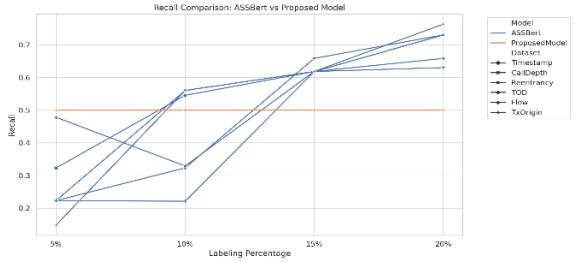

Figure 4: Recall comparison

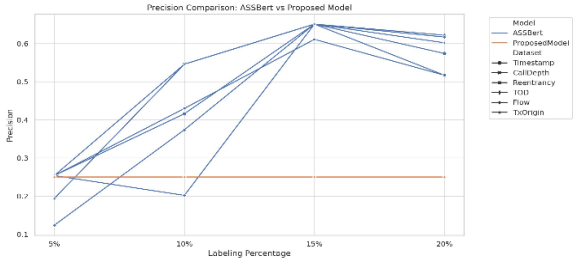

Figure 5: Precision comparison

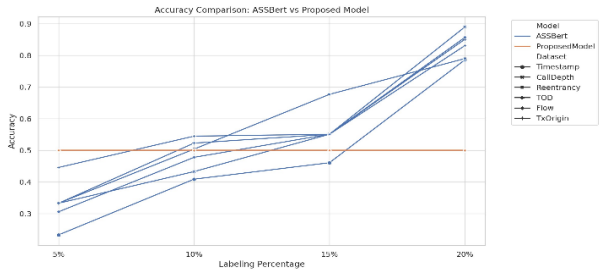

Figure 6: Accuracy comparison

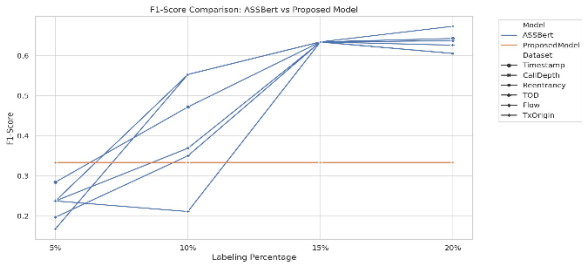

Figure 7: F1-Score comparison

Conversely, the recall performance of the Proposed Model (0.500) remains stable and frequently exceeds that of ASSBert, particularly at lower labeling percentages. ASSBert, in contrast, is initialized with significantly lower recall (e.g., 0.223 for CallDepth and Reentrancy at 5%). However, its performance demonstrates a strong positive correlation with labeling percentage, eventually rivaling that of the Proposed Model at higher label levels. The superior recall of the Proposed Model in low-label settings indicates a greater efficacy in identifying true positives even when trained on limited data. This characteristic renders it particularly suitable for security-critical tasks like smart contract vulnerability detection, where maximizing the identification of potential vulnerabilities is prioritized over minimizing false alarms. In summary, the Proposed Model excels in recall and early-stage learning, demonstrating significant promise for low-resource scenarios. However, it lags in precision and lacks the scalability of ASSBert, which more effectively capitalizes on increased label availability. These results position the Proposed Model as a robust baseline for rapid deployment where annotations are scarce, while ASSBert represents a more scalable solution as labeling efforts intensify. Enhancing the precision of the Proposed Model remains a key direction for increasing its competitiveness in real-world applications.

CONCLUSION

This study presented a novel ELECTRA-based model for smart contract vulnerability detection, addressing the limitations of traditional static/dynamic analysis and prior machine learning approaches such as ASSBERT. The proposed model leverages context-aware masking and bidirectional transformer-based pretraining to enhance the detection of logic-based vulnerabilities in Solidity smart contracts. The comparative evaluation against ASSBERT across six benchmark datasets and varying labeling percentages revealed several key findings. Notably, the proposed model demonstrated superior recall and consistent accuracy, particularly in low-labeling scenarios (5%–10%), where traditional models like ASSBERT underperformed. This highlights the model's potential for practical deployment in environments with limited annotated data. While ASSBERT showed improved performance at higher labeling percentages, especially in terms of precision and overall accuracy, the proposed model maintained stable performance across all settings. The consistent F1-score (0.333) and recall (0.500) suggest a bias toward identifying true positives, which is critical in security-sensitive contexts such as smart contract auditing. Despite these strengths, the model's lower precision (0.250) indicates a higher rate of false positives, which could lead to unnecessary interventions or false alerts. Additionally, the modest gain in validation loss over four epochs (from 0.689 to 0.684) suggests that the model is in its early stages of optimization and may not yet have reached its full potential.

Future work will focus on several key directions. First, enhancing the masking strategy to capture more semantic and syntactic contract structures may help reduce false positives and improve precision. Second, increasing training epochs and expanding the dataset will likely enhance generalization and metric convergence. Additionally, integrating graph-based contract representations, contrastive learning, or prompt-based fine-tuning could further enrich the model's contextual understanding. Finally, incorporating automated patch suggestion mechanisms in conjunction with detection will offer a more comprehensive solution for smart contract security. In conclusion, this research provides an effective foundation for vulnerability detection in blockchain environments using ELECTRA, with promising directions for refinement and real-world deployment.

ACKNOWLEDGMENT

The research work was sponsored by the tertiary Education Trust Fund (TETFund) Nigeria.

REFERENCES

Aburass, S., Dorgham, O., & Rumman, M. A. (2024). An Ensemble Approach to Question Classification: Integrating Electra Transformer, GloVe, and LSTM. *International Journal of Advanced Computer Science and Applications*, *15*(1). https://doi.org/10.14569/ijacsa.2024.0150148

Aladhadh, S., Alwabli, H., Moulahi, T., & Al Asqah, M. (2022). BChainGuard: A New Framework for Cyberthreats Detection in Blockchain Using Machine Learning. *Applied Sciences*, *12*(23), 12026. https://doi.org/10.3390/app122312026

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lucic, M., & Schmid, C. (2021). ViViT: A Video Vision Transformer. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 6816–6826. https://doi.org/10.1109/iccv48922.2021.00676

Brauwers, G., & Frasincar, F. (2023). A General Survey on Attention Mechanisms in Deep Learning. *IEEE Transactions on Knowledge and Data Engineering*, *35*(4), 3279–3298. https://doi.org/10.1109/TKDE.2021.3126456

Bu, J., Li, W., Li, Z., Zhang, Z., & Li, X. (2025). *SmartBugBert: BERT-Enhanced Vulnerability Detection for Smart Contract Bytecode* (No. arXiv:2504.05002). arXiv. https://doi.org/10.48550/arXiv.2504.05002

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., Zhang, P., Dong, H., Xiao, Y., Ji, S., & Li, W. (2023). A survey on smart contract vulnerabilities: Data sources, detection and repair. *Information and Software Technology*, *159*, 107221. https://doi.org/10.1016/j.infsof.2023.107221

Clark, K., Luong, M.-T., & Le, Q. V. (2020). *ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.

Fei, J., Chen, X., & Zhao, X. (2023). MSmart: Smart contract vulnerability analysis and improved strategies based on smartcheck. *Applied Sciences*, *13*(3), 1733.

Ferreira, J. F., Cruz, P., Durieux, T., & Abreu, R. (2020). *Smartbugs: A framework to analyze solidity smart contracts*. 1349–1352.

Galal, O., Abdel-Gawad, A. H., & Farouk, M. (2024). Rethinking of BERT sentence embedding for text classification. *Neural Computing and Applications*, *36*(32), 20245–20258. https://doi.org/10.1007/s00521-024-10212-3

Ivanov, N., Li, C., Yan, Q., Sun, Z., Cao, Z., & Luo, X. (2023). Security Defense For Smart Contracts: A Comprehensive Survey. *ACM Computing Surveys*, *55*(14s), 1–37. https://doi.org/10.1145/3593293

Mi, F., Zhao, C., Wang, Z., Halim, S. M., Li, X., Wu, Z., Khan, L., & Thuraisingham, B. (2023). *An Automated Vulnerability Detection Framework for Smart Contracts* (No. arXiv:2301.08824). arXiv. https://doi.org/10.48550/arXiv.2301.08824

Pham Trong Linh, & Minh Thanh, T. (2023). Proposing of Imaging Graph Neural Network with Defined Security Pattern for Improving Smart Contract Vulnerability Detection. *Research and Development on Information and Communication Technology*, 70–79. https://doi.org/10.32913/mic-ict-research.v2023.n2.1198

Singh, S., & Mahmood, A. (2021). The NLP Cookbook: Modern Recipes for Transformer Based Deep Learning Architectures. *IEEE Access*, *9*, 68675–68702. https://doi.org/10.1109/access.2021.3077350

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

Tang, X., Du, Y., Lai, A., Zhang, Z., & Shi, L. (2023). Deep learning-based solution for smart contract vulnerabilities detection. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-023-47219-0

Usman, U. B., Umar, K., & Agaie, A. I. (2024). CodeELECTRA: An ELECTRA-based approach for improved vulnerability detection in blockchain smart contracts. *Dutse Journal of Pure and Applied Sciences*, *10*(3b), 95–105. https://doi.org/10.4314/dujopas.v10i3b.11

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (No. arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762

Vidal, F. R., Ivaki, N., & Laranjeiro, N. (2024). OpenSCV: An Open Hierarchical Taxonomy for Smart Contract Vulnerabilities. *Empirical Software Engineering*, *29*(4). https://doi.org/10.1007/s10664-024-10446-8

Yuan, X., Lin, G., Tai, Y., & Zhang, J. (2022). Deep Neural Embedding for Software Vulnerability Discovery: Comparison and Optimization. *Security and Communication Networks*, *2022*, 1–12. https://doi.org/10.1155/2022/5203217