iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.84002

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

Smart Contracts Vulnerability Detection Using Machine Learning and Large Language Models

Rehana Qudsiya[1], O. B. V. Ramanaiah[2]
[1]*M. Tech Student of Department of Computer Science and Engineering, JNTU Hyderabad, Telangana, India*
[2] *Senior Professor of the Department of Computer Science and Engineering, JNTU Hyderabad, Telangana, India*

Abstract: ***As blockchain technology and smart contracts gain widespread adoption, ensuring their security is essential to prevent financial and operational risks. Detecting vulnerabilities in smart contracts using automated techniques provides a reliable and scalable solution. This study utilizes the Smart Contract Vulnerabilities Dataset from Kaggle, containing annotated smart contracts with labeled vulnerabilities. Preprocessing includes tokenization and exploratory data analysis to extract meaningful textual patterns. Deep learning models such as LSTM and BERT are trained and evaluated using accuracy, precision, recall, and F1-score. To further improve detection performance, BERT embeddings are combined with BiLSTM and CNN + LSTM architectures. A Flask-based user interface enables real-time vulnerability prediction. Experimental results show that the CNN + LSTM model outperforms all other models, achieving 95 percent accuracy and demonstrating strong capability in identifying smart contract vulnerabilities.***

Keywords: Smart Contract Vulnerability Detection, BERT, LSTM, CNN-LSTM, Performance Analysis

I. INTRODUCTION

The rapid evolution of blockchain technology has revolutionized digital transactions and decentralized ecosystems, with smart contracts serving as one of its most transformative components. Smart contracts self-executing programs that automatically enforce contractual terms have been widely adopted across diverse domains such as decentralized finance (DeFi), healthcare, logistics, and supply chain management [1]. Their ability to eliminate intermediaries and ensure transparent, tamper-proof execution has made them indispensable to modern blockchain infrastructures. As the scale and complexity of blockchain based applications expand, the security of smart contracts has emerged as a fundamental concern. Even minor coding errors or overlooked logical flaws can result in severe vulnerabilities, leading to substantial financial losses and reputational damage to decentralized platforms [2]. Consequently, developing reliable, intelligent, and automated methods for identifying vulnerabilities within smart contracts has become a critical research priority.

Despite the proliferation of research on blockchain security, existing smart contract vulnerability detection approaches remain limited in accuracy and adaptability. Traditional static and symbolic analysis tools have been effective in identifying predefined vulnerability patterns but struggle with detecting unseen or context-dependent flaws [3]. Furthermore, these rule-based systems are often constrained by the need for extensive manual configuration and predefined vulnerability signatures [4]. As smart contracts continue to evolve in both structure and functionality, conventional detection methods fail to generalize across diverse contract types and programming behaviors [5]. This creates a critical gap in automated vulnerability analysis—specifically, the need for models capable of understanding complex, context-aware code semantics and dynamically identifying potential risks beyond surface-level patterns. The inability of current systems to capture deep syntactic and semantic dependencies within smart contract code underscores the urgent need for innovative solutions leveraging more advanced computational intelligence [6].

The primary objective of this research is to explore and evaluate the effectiveness of integrating large language models (LLMs) and machine learning frameworks for enhancing smart contract vulnerability detection. By leveraging the contextual understanding and pattern recognition capabilities of language models trained on extensive code corpora, this study seeks to improve the interpretability, scalability, and precision of vulnerability identification [7]. The research emphasizes a comparative evaluation of different computational paradigms to determine their capability in capturing intricate security flaws within smart contract code. It further contributes by establishing comprehensive benchmarks that support the assessment of various model architectures and by examining how domain-specific fine-tuning can enhance the robustness and generalization of automated detection frameworks [8].

The significance of this study lies in its potential to strengthen blockchain security and trustworthiness by enabling proactive and accurate detection of vulnerabilities before deployment.

©IJRASET: All Rights are Reserved

Enhanced vulnerability detection mechanisms can prevent large scale financial exploitation, promote sustainable blockchain adoption, and support the development of more secure decentralized ecosystems [9]. Furthermore, by providing an empirical foundation for future integration of artificial intelligence and blockchain, this work opens avenues for the design of intelligent auditing systems that ensure transparency, reliability, and compliance in smart contract execution [10]. Ultimately, this research contributes toward establishing a secure, scalable, and intelligent framework for safeguarding the integrity of blockchain based systems.

II. LITERATURE REVIEW

The security of Ethereum smart contracts has been a subject of sustained research interest, driven by high-profile exploits targeting Reentrancy, Integer Overflow/Underflow, Timestamp Dependency, and Dangerous Delegatecall vulnerabilities. Sayeed et al. [11] provided a foundational taxonomy of smart contract attack vectors and protection strategies on Ethereum, demonstrating that conventional static analyzers and rule-based tools, while effective against known patterns, generate substantial false positives and fail to reason about context-dependent execution logic. To overcome these limitations, Qian et al. [12] proposed sequential LSTM-based models that trace contract execution flow for automated Reentrancy detection, achieving measurably lower false-positive rates and improved accuracy over static approaches on publicly labeled datasets.

Building on recurrent architectures, subsequent studies adopted broader deep learning frameworks for multi-class vulnerability classification. Prifti et al. [13] applied CNN, LSTM, BiLSTM, and GRU networks directly on EVM bytecode, showing that convolutional layers extract local structural patterns while bidirectional recurrent layers capture long-range sequential dependencies inherent in Reentrancy and Integer Overflow scenarios. Fan et al. [14] addressed the persistent challenge of limited labeled data through a multi-layer feature fusion strategy that combines syntactic, structural, and semantic contract representations, yielding improved generalization under small-sample conditions. Despite these advances, deep models lacked the ability to encode high-level program semantics, motivating the adoption of transformer-based architectures.

The integration of large language models marked a significant leap in detection capability. Hu et al. [15] demonstrated that LLMs prompted with domain-specific vulnerability knowledge outperform both static analyzers and ML models in identifying subtle, logic-level flaws. Choi et al. [16] further enriched LLM inputs by incorporating program dependency graphs, combining structural code analysis with transformer-based semantic embeddings to achieve superior multi-class detection performance. He et al. [17] showed that fine-tuning pre-trained language models on Solidity corpora substantially improves classification accuracy and reduces the need for manual feature engineering, while Hossain et al. [18] proposed a complementary pipeline in which LLM-generated embeddings are consumed by lightweight ML classifiers, balancing contextual richness with computational tractability. Finally, Zaazaa and El Bakkali [19] introduced SmartLLMSentry, a comprehensive LLM-based auditing framework integrating multiple detection strategies within a unified pipeline, demonstrating broad vulnerability coverage across all major contract weakness categories.

Collectively, this body of work reveals persistent research gaps: most approaches address only one or two vulnerability classes rather than simultaneous multi-class classification [11]–[13]; class imbalance across vulnerability types degrades detection of minority classes [14]; and fully fine-tuned LLM frameworks impose high computational overhead that limits practical deployment [15]–[19]. Additionally, limited model interpretability across methods constrains their adoption in security-critical settings. These gaps motivate the present work, which integrates BERT-based contextual embeddings with sequential and convolutional deep learning models encompassing standalone LSTM, BERT+BiLSTM, and LSTM+CNN architectures to deliver accurate, scalable, and comprehensive multi-class vulnerability detection for Ethereum smart contracts.

III. PROPOSED METHODOLOGY

The proposed system focuses on automated detection of vulnerabilities in smart contracts using the Smart Contract Vulnerabilities Dataset obtained from Kaggle, which contains annotated contract code with labeled security flaws.

A. Overview Of Proposed System

The overall workflow involves systematic preprocessing and feature representation to enable effective learning of syntactic, semantic, and contextual patterns within smart contract code.

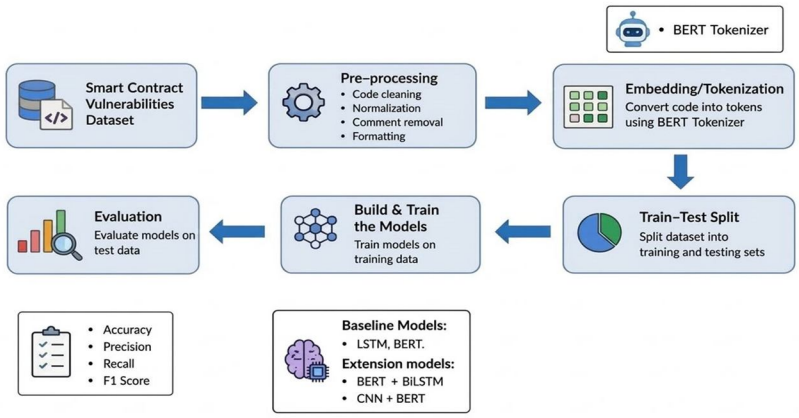


Fig. 1 System Architecture

The Smart Contract Vulnerabilities Dataset is collected, comprising Solidity smart contract source codes containing different vulnerability categories. The collected dataset serves as the input to the proposed system.

In the Pre-processing stage, the raw smart contract code undergoes several cleaning operations, including code cleaning, normalization, comment removal, and formatting. These preprocessing steps eliminate unnecessary information and ensure that the source code is transformed into a consistent format suitable for model training.

After preprocessing, the cleaned smart contract code is subjected to Embedding/Tokenization using the pretrained BERT tokenizer. The tokenizer converts the textual source code into numerical token representations while preserving contextual information. This transformation enables deep learning models to effectively process the source code.

Subsequently, the tokenized dataset is divided into training and testing sets through the Train-Test Split process. The training set is utilized for model learning, whereas the testing set is reserved for performance evaluation.

In the Build and Train Models phase, multiple vulnerability detection models are developed and trained. The study includes baseline models such as LSTM and BERT, along with hybrid architectures including BERT + BiLSTM and CNN + BERT. These models learn vulnerability-specific patterns from the training data and generate predictions for unseen smart contract samples.

Finally, the trained models are evaluated on the test dataset using standard performance metrics, namely Accuracy, Precision, Recall, and F1-Score. These metrics provide a comprehensive assessment of the effectiveness of each model in detecting and classifying smart contract vulnerabilities.

B. Dataset Description

A labeled dataset of Solidity smart contracts is used for vulnerability detection. The dataset contains 2,217 smart contract samples belonging to four vulnerability classes: Reentrancy, Integer Overflow/Underflow, Timestamp Dependency, and Dangerous Delegatecall. Each contract is assigned a vulnerability label, enabling supervised learning for classification. The dataset is imbalanced, with Reentrancy having the highest number of samples and Dangerous Delegatecall having the lowest, reflecting real-world vulnerability distributions in Ethereum smart contracts.

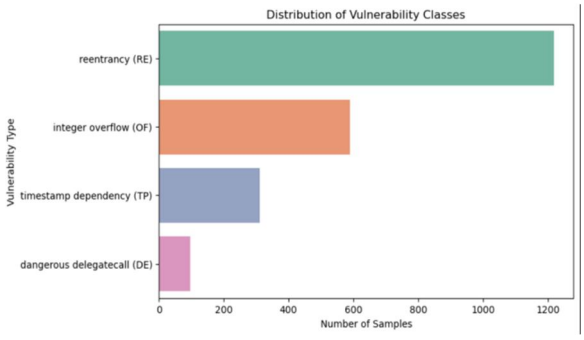


Fig. 2 Smart Contract Vulnerabilities Dataset

C. Data Pre-processing

The preprocessing stage converts raw smart contract data into a clean and consistent format. Missing records are handled, vulnerability labels are standardized and encoded, and the source code is cleaned by removing comments, extra spaces, and unnecessary content. These steps improve data quality and help the models learn vulnerability patterns more effectively.

D. Feature Extraction and Tokenization

Figure 3 illustrates the tokenization and embedding process used for smart contract vulnerability detection. First, the preprocessed Solidity smart contract code is provided as input. The BERT tokenizer then splits the code into smaller meaningful tokens such as keywords, operators, and function names. These tokens are converted into token IDs, which represent the code in numerical form. Attention masks are generated to distinguish valid tokens from padded values. Next, BERT creates contextual embeddings that capture the semantic relationships between tokens. Finally, the token IDs, attention masks, and embeddings are supplied to deep learning models such as BERT, BERT+BiLSTM and LSTM+CNN for vulnerability prediction and classification.

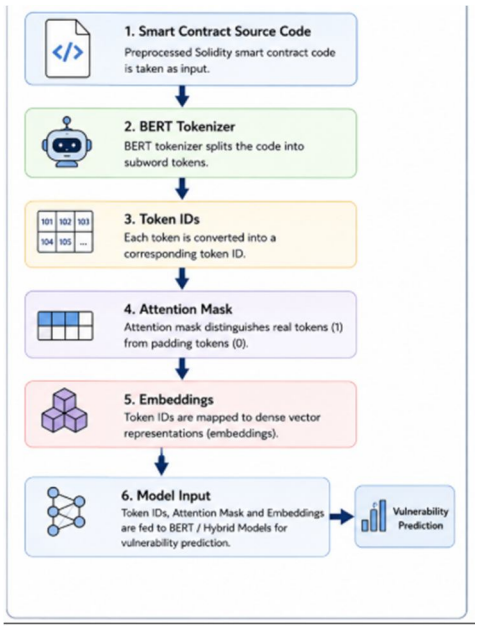


Fig 1 Feature Extraction And Tokenization

E. Algorithms

1) Long Short-Term Memory (LSTM): Long Short-Term Memory (LSTM) is a recurrent neural network architecture designed to capture long-range dependencies in sequential data, enhancing model performance on tasks involving order-sensitive patterns. It employs gated mechanisms to selectively retain, update, or discard information, thereby preserving crucial temporal context and mitigating gradient vanishing issues. By modeling sequential relationships effectively, LSTM improves classification accuracy, robustness, and generalization across data exhibiting complex temporal or contextual interactions, making it particularly suitable for tasks requiring dynamic pattern recognition over extended sequences.

$$h_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o) \cdot \tanh(C_t) \quad (1)$$

2) Bidirectional Encoder Representations from Transformers (BERT): BERT is a transformer-based architecture that utilizes bidirectional attention mechanisms to model deep contextual relationships between tokens. By simultaneously analyzing both left and right contexts, it generates rich semantic embeddings that capture syntactic and semantic dependencies effectively. Pre-training on large-scale corpora followed by task-specific fine-tuning enables superior generalization and adaptability. BERT [30] enhances performance and interpretability by detecting nuanced patterns and subtle contextual cues, improving classification precision and overall robustness in tasks involving complex textual or structured sequences.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

3) CNN + LSTM: The CNN + LSTM hybrid architecture combines the spatial feature extraction capability of convolutional networks with the sequential learning strength of LSTMs. CNN layers capture hierarchical and local patterns, while LSTM layers model temporal dependencies across sequences. This integration enhances representation learning by jointly capturing structural and temporal relationships, improving detection accuracy, generalization, and robustness. The architecture is particularly effective in scenarios requiring comprehension of complex spatial-temporal interactions, ensuring dynamic pattern recognition and reliable performance across varied sequential or structured data.

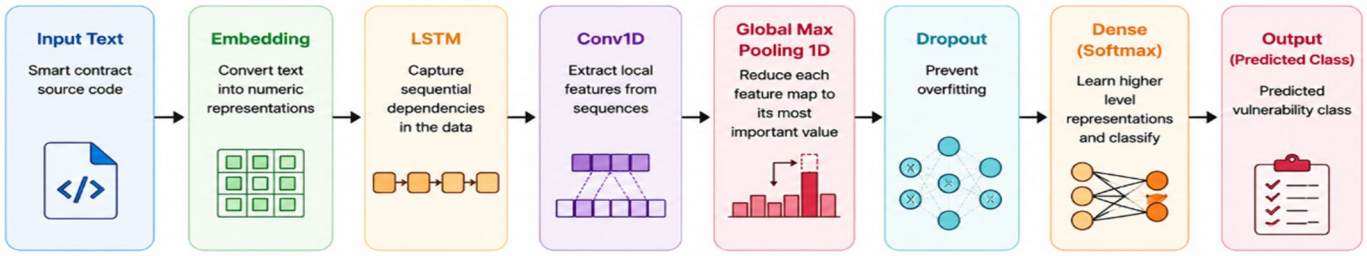


Fig 4: CNN+LSTM Architecture

4) BERT + BiLSTM: The BERT + BiLSTM architecture integrates BERT's bidirectional contextual embeddings with Bidirectional Long Short-Term Memory networks to enhance temporal and contextual understanding. BERT encodes rich semantic features, while BiLSTM processes them in both forward and backward directions, capturing intricate dependencies within sequences. This combination strengthens feature representation, improving classification accuracy, robustness, and interpretability. The architecture is well-suited for complex sequential data, as it leverages global semantic understanding alongside fine-grained temporal modeling to identify subtle and context-dependent patterns efficiently.

*F. Experimental **setup***

The experiments were conducted on a labeled dataset containing 2,217 Solidity smart contract samples belonging to four vulnerability classes: Reentrancy, Integer Overflow/Underflow, Timestamp Dependency, and Dangerous Delegatecall. The smart contract source code was preprocessed through code cleaning, normalization, comment removal, and label encoding. The cleaned code was then tokenized using the BERT tokenizer to generate token IDs, attention masks, and contextual embeddings.

The dataset was divided into training and testing sets using an 80:20 split. Five deep learning and hybrid models, namely LSTM, BERT, BERT+BiLSTM and LSTM+CNN, were trained and evaluated for vulnerability classification. The models were implemented using Python with TensorFlow, Keras, and the Hugging Face Transformers library.

Performance was evaluated using Accuracy, Precision, Recall, and F1-Score. These metrics were selected to assess the effectiveness of each model in identifying vulnerabilities and handling class imbalance within the dataset. A comparative analysis was performed to determine the most suitable model for smart contract vulnerability detection.

IV. EXPERIMENTAL RESULTS

The experimental results indicate that the LSTM + CNN model outperformed all other evaluated models, achieving the highest accuracy of 95.05% along with superior precision, recall, and F1-score values. The standalone LSTM model also exhibited strong performance with an accuracy of 93.69%. In comparison, the BERT and BERT + BiLSTM models achieved lower performance, with accuracies of 80.63% and 82.43%, respectively. The improved performance of the LSTM + CNN architecture can be attributed to its ability to effectively capture both sequential dependencies and local patterns within smart contract source code. These findings demonstrate the effectiveness of hybrid deep learning architectures for smart contract vulnerability detection.

Table.1 Performance Evaluation Table

Model	Accuracy	Precision	Recall	F1-Score
LSTM	0.9369	0.9380	0.9369	0.9370
BERT	0.8063	0.8113	0.8063	0.8078
BERT + BiLSTM	0.8243	0.8222	0.8243	0.8204
LSTM + CNN	0.9505	0.9506	0.9505	0.9501

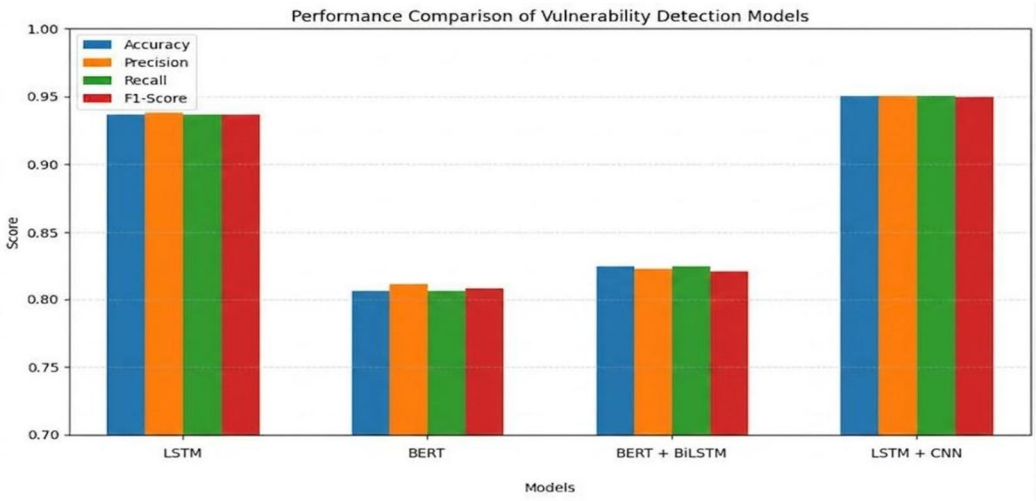


Fig. 5: Overall Performance Comparison of Models

Figure 5 presents the comparative performance of the evaluated models in terms of Accuracy, Precision, Recall, and F1-Score. The LSTM + CNN model achieved the highest scores across all metrics, demonstrating its superior capability in detecting smart contract vulnerabilities. The standalone LSTM model also exhibited strong performance, whereas the BERT and BERT + BiLSTM models achieved comparatively lower results. Overall, the graph indicates that the hybrid LSTM + CNN architecture is the most effective model among the evaluated approaches.

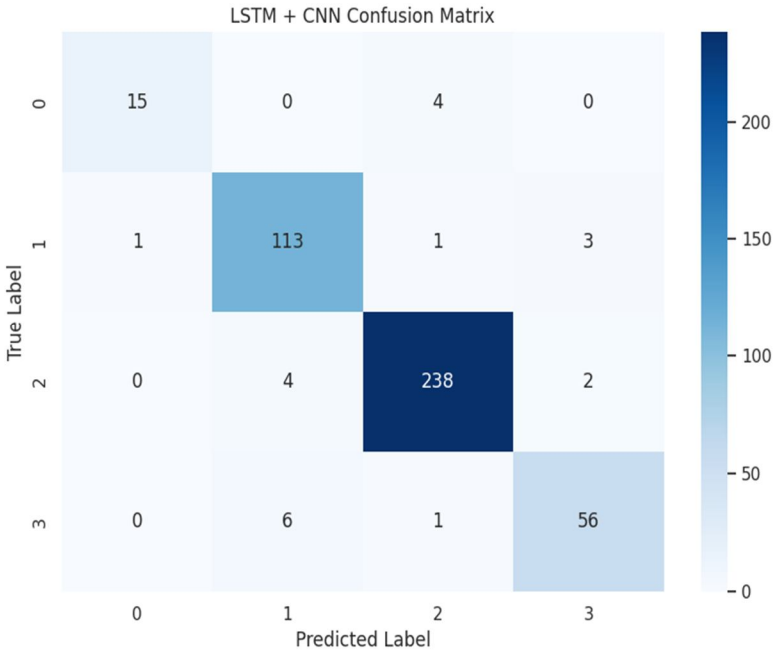


Fig. 6: LSTM+CNN Model Confusion Matrix

Figure 6 illustrates the confusion matrix of the LSTM + CNN model on the test dataset. A high concentration of values along the diagonal indicates that the model correctly classified most vulnerability instances. Specifically, the model accurately identified 15 samples of class 0, 113 samples of class 1, 238 samples of class 2, and 56 samples of class 3. Only a small number of samples were misclassified among different classes, demonstrating the model's strong capability.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

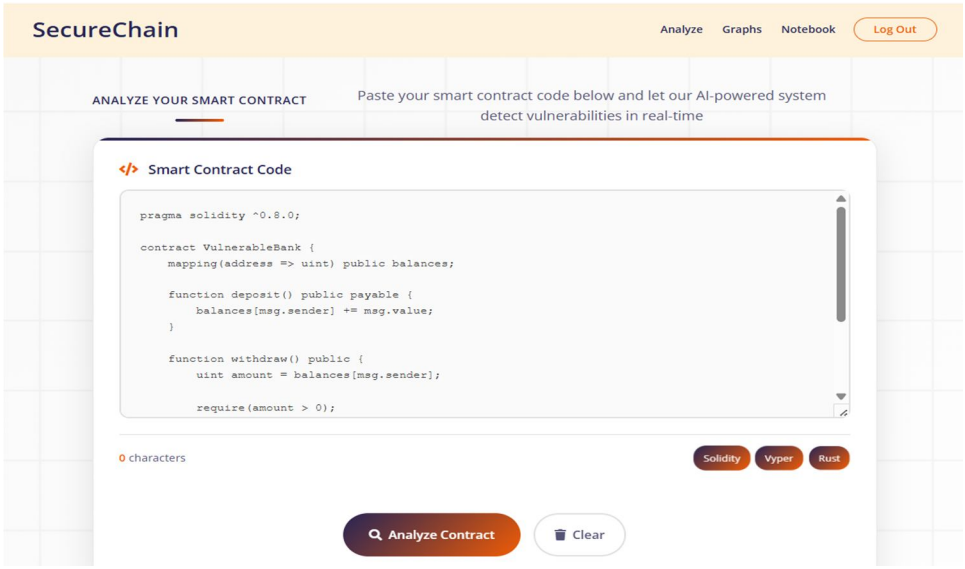


Fig 7: Smart Contract Code Analysis Interface

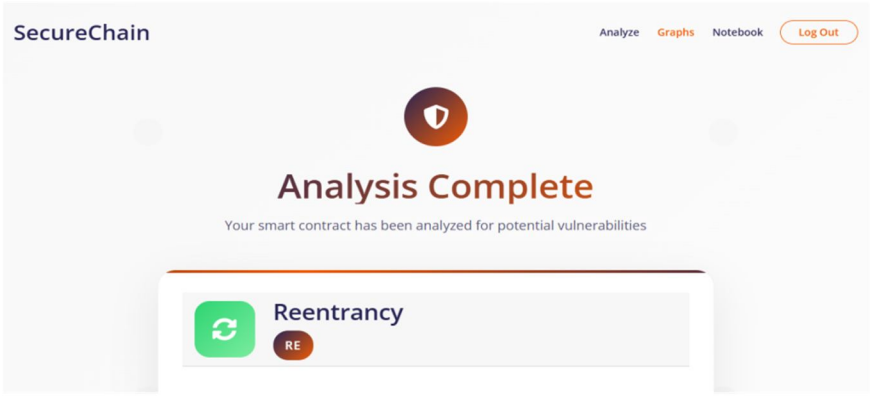


Fig 8:Output of the Proposed Vulnerability Detection System

Figures 7 and 8 depict the graphical user interface and output of the proposed smart contract vulnerability detection system. The system enables users to input Solidity smart contract code through a web-based interface and perform automated vulnerability analysis. After processing the submitted code, the system predicts the vulnerability type and displays the analysis result. In the illustrated example, the proposed model successfully detects a Reentrancy vulnerability, demonstrating the practical applicability of the developed framework.

V. CONCLUSION AND FUTURE WORK

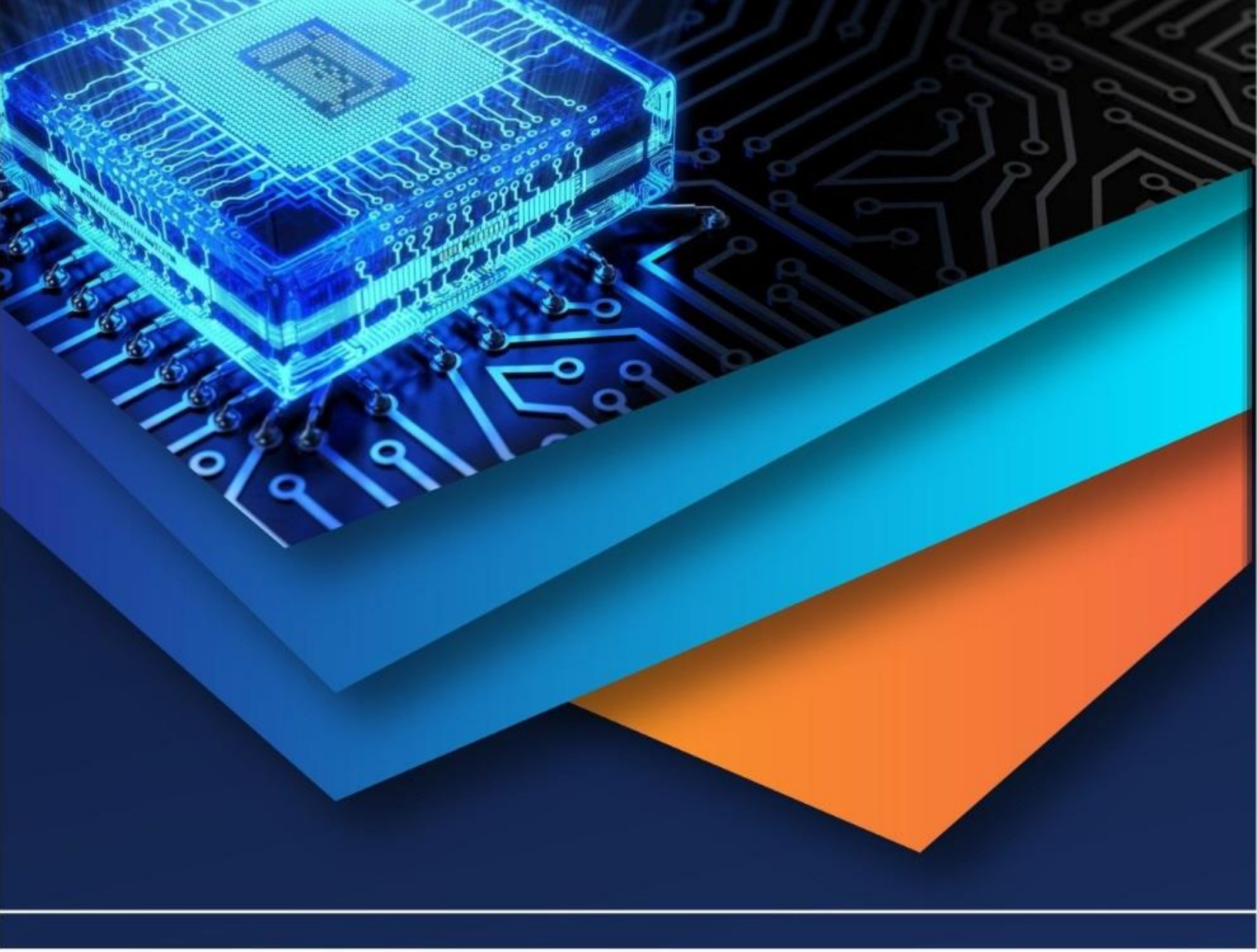
This paper presented a smart contract vulnerability detection framework based on deep learning and transformer-based models for identifying Reentrancy, Integer Overflow/Underflow, Timestamp Dependency, and Dangerous Delegatecall vulnerabilities. The proposed approach included data preprocessing, BERT-based tokenization, feature extraction, and vulnerability classification. Experimental results demonstrated that all evaluated models were capable of detecting vulnerability patterns in smart contract code. Among them, the LSTM+CNN model achieved the best performance with an accuracy of 95.05%, indicating its effectiveness in capturing both sequential and structural features of smart contracts.

Future work will focus on extending the framework to support additional vulnerability types and larger smart contract datasets. The integration of advanced transformer architectures and code-specific language models may further improve detection accuracy. Additionally, techniques such as data augmentation and class balancing can be explored to enhance the classification performance of minority vulnerability classes.

REFERENCES

[1] Yuan, H., Yu, L., Huang, Z., Zhang, J., Lu, J., Cheng, S., ... & Zuo, C. (2025). Mos: Towards effective smart contract vulnerability detection through mixture-of-experts tuning of large language models. arXiv preprint arXiv:2504.12234.
[2] Yu, L., Huang, Z., Yuan, H., Cheng, S., Yang, L., Zhang, F., ... & Zuo, C. (2025). Smart-LLaMA-DPO: Reinforced Large Language Model for Explainable Smart Contract Vulnerability Detection. Proceedings of the ACM on Software Engineering, 2(ISSTA), 182-205.
[3] Luo, Y., Xu, W., Andersson, K., Hossain, M. S., & Xu, D. (2024, August). Fellmvp: An ensemble llm framework for classifying smart contract vulnerabilities. In 2024 IEEE International Conference on Blockchain (Blockchain) (pp. 89-96). IEEE.
[4] He, F., Li, F., & Liang, P. (2024). Enhancing smart contract security: Leveraging pre-trained language models for advanced vulnerability detection. IET blockchain, 4, 543-554.
[5] Yu, L., Chen, S., Yuan, H., Wang, P., Huang, Z., Zhang, J., ... & Ma, J. (2024). Smart-LLaMA: two-stage post-training of large language models for smart contract vulnerability detection and explanation. arXiv preprint arXiv:2411.06221.
[6] Kiani, R., & Sheng, V. S. (2024). Ethereum smart contract vulnerability detection and machine learning-driven solutions: A systematic literature review. Electronics, 13(12), 2295. https://www.mdpi.com/2079-9292/13/12/2295
[7] Kim, J., Lee, S., & Kim, H. (2024). Robust vulnerability detection in solidity-based ethereum smart contracts using fine-tuned transformer encoder models. IEEE Access. https://doi.org/10.1109/ACCESS.2024.3482389
[8] Boi, B., Esposito, C., & Lee, S. (2024). Smart contract vulnerability detection: The role of large language model (LLM). ACM SIGAPP Applied Computing Review, 24(2), 19–29. https://doi.org/10.1145/3687251.3687253
[9] Ma, W., Wu, D., Sun, Y., Wang, T., Liu, S., Zhang, J., Xue, Y., & Liu, Y. (2024). Combining fine-tuning and LLM-based agents for intuitive smart contract auditing with justifications. arXiv preprint arXiv: 2403.16073. https://arxiv.org/abs/2403.16073
[10] Prifti, L., Çiço, B., & Karras, D. A. (2024). Smart contract vulnerability detection using deep learning algorithms on EVM bytecode. In 2024 13th Mediterranean Conference on Embedded Computing (MECO). IEEE. https://doi.org/10.1109/MECO62516.2024.10577852
[11] S. Sayeed, H. Marco-Gisbert and T. Caira, "Smart Contract: Attacks and Protections," IEEE Access, vol. 8, pp. 24416–24427, 2020, doi: 10.1109/ACCESS.2020.2970495.
[12] P. Qian, Z. Liu, Q. He, R. Zimmermann and X. Wang, "Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models," IEEE Access, vol. 8, pp. 19685–19695, 2020, doi: 10.1109/ACCESS.2020.2969429.
[13] L. Prifti, B. Cico and D. Karras, "Smart Contract Vulnerability Detection Using Deep Learning Algorithms on EVM Bytecode," MECO, 2024, pp. 1–7, doi: 10.1109/MECO62516.2024.10577852.
[14] J. Fan, Y. He, and H. Wu, "Small Sample Smart Contract Vulnerability Detection Based on Multi-Layer Feature Fusion," Complex & Intelligent Systems, vol. 11, Art. no. 198, 2025, doi: 10.1007/s40747-025-01782-3.
[15] S. Hu et al., "Large Language Model-Powered Smart Contract Vulnerability Detection: New Perspectives," TPS-ISA, 2023, pp. 297–306, doi: 10.1109/TPS-ISA58951.2023.00044.
[16] R.-Y. Choi et al., "Smart Contract Vulnerability Detection Using Large Language Models and Graph Structural Analysis," Computers, Materials & Continua, vol. 83, no. 1, pp. 785–801, 2025, doi: 10.32604/cmc.2025.061185.
[17] F. He, F. Li, and P. Liang, "Enhancing Smart Contract Security: Leveraging Pre-Trained Language Models for Advanced Vulnerability Detection," IET Blockchain, vol. 4, pp. 543–554, 2024, doi: 10.1049/blc2.12072.
[18] S. M. M. Hossain, A. Altarawneh and J. Roberts, "Leveraging Large Language Models and Machine Learning for Smart Contract Vulnerability Detection," CCWC, 2025, pp. 577–583, doi: 10.1109/CCWC62904.2025.10903869.
[19] O. Zaazaa and H. El Bakkali, "SmartLLMSentry: A Comprehensive LLM Based Smart Contract Vulnerability Detection Framework," Journal of Metaverse, vol. 4, pp. 126–137, 2024, doi: 10.57019/jmv.1489060.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)