"Advancing Next-Generation Smart Contracts with Real-Time Threat Detection and Vulnerability Testing"

Sangeetha R
Assistant Professor
CIST, ManasaGangortri, University of Mysore
Mysuru-570006
Orcid Id:0009-0004-1452-0549

Dr. Veena M N
Professor
Department of MCA
PES college of Engineering, Shankar Gowda road, Mandya-571401
Orcid Id:0009-0004-5953-1065

Abstract:

Smart contract vulnerabilities have gone forth as a major cause of threatening the transaction security of block-chain. Smart contracts are widely used in industries like finance and the Internet of Things (IoT) and are essential to the block-chain attribute. The state-of-the-art research methods rely on deep learning to mitigate this threat. These treat if each input contract as an independent entity feed it into a deep learning model to learn vulnerability patterns by fitting vulnerability labels. This paper aims to address the issues present in existing methods for smart contract classification and proposes a smart contract classification algorithm called modified model for Cluster-BERT, based on convolution neural network CNN techniques. The model reduces the computational burden of self-attention mechanisms by clustering attention heads, thereby improving training efficiency. The Cluster-BERT model comprises multiple modules. Initially the pre-process smart contract data, converting abstract syntax trees and graph structure features into text representations in the next phase the core model introduces neural clustering methods to reduce computational complexity at last the model by finding the optimal number of centroids, achieving a balance between training efficiency and classification accuracy. The experimental results show that our proposed Cluster-BERT achieved better accuracy, a recall, and an F1 score compared to the existing prototype, which indicates a noticeable improvement over the baseline model. Further the proposed model reduces computational complexity from quadratic to linear, resulting in an average reduction in training time and prediction time compared to the baseline model.

I. INTRODUCTION

Smart contracts are an essential part of block-chain technology and are crucial for developing decentralized applications. Smart contracts, which function as decentralized applications on the block-chain, are designed using various block-chain-specific features, including the gas mechanism, delegate call mechanism, exception passing mechanism, and other unique mechanisms specific to smart contracts [1]. While these characteristics have supported the swift adoption and evolution of smart contracts on the block-chain, the presence of these specific processes has also led to various vulnerabilities in numerous smart contracts that have emerged from them. However, existing methods suffer from inefficiencies and high computational complexity when dealing with smart contract data. smart contract vulnerabilities can cause financial losses and system crashes. Static analysis tools are frequently used to detect vulnerabilities in smart contracts, but they often result in false positives and false negatives because of their high reliance on predefined rules and lack of semantic analysis capabilities. Modern contracts can be revised to address vulnerabilities, but the costs associated with frequent updates can be challenging, and the repercussions of a vulnerability attack are irreparable. This highlights the need for a comprehensive security assessment of the smart contracts before deployment, aiming to prevent the distribution of problematic contracts [2, 3]. Current methods for detecting vulnerabilities in smart contracts can be generally categorized into two groups: traditional detection techniques that depend on expert rules derived from practical experience and are enhanced by various automated vulnerability detection tools, and methods based on deep learning for detecting vulnerabilities [4]. The methods for detecting vulnerabilities in smart contracts encounter numerous challenges, including low efficiency, a restricted range of vulnerability types, and the absence of standardized datasets appropriate for machine learning applications. A Bayesian network framework was developed to evaluate the severity of various vulnerabilities and pinpoint their origins. However, it can only identify a limited range of vulnerabilities, and the accuracy is somewhat low. This paper proposes a research method based on machine learning algorithms to address the problems mentioned above. Compared with traditional static and dynamic vulnerability detection methods, machine learning-based methods can significantly reduce the reliance on mathematical knowledge, detect vulnerabilities with higher accuracy, and offer good scalability [5].

The proposed organization of the Infra data is then prioritized for Application Scans using methods to identify security vulnerabilities in web applications and source code by automated front-end scanning or source code analysis is as shown in Figure 1.

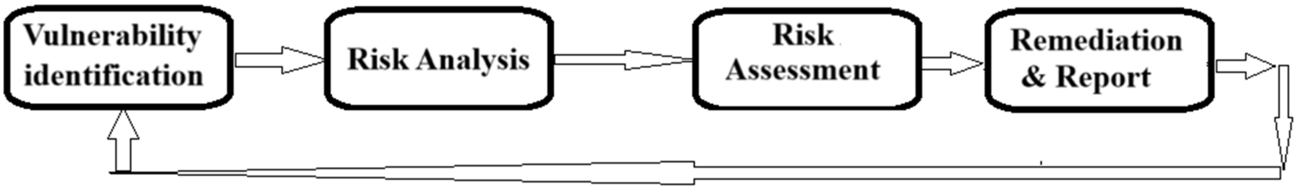


Figure 1: Proposed Vulnerability Scanning

The Threat, Vulnerability, and Risk Assessment program helps to understand how proposed system identifies and mitigates the impact of physical and environmental threats to data centers is as shown in Figure 1. Proposed system is committed to continually updating its risk assessments and methodologies for improvements and as conditions change. Vulnerability scanning works by systematically examining computer systems, networks, and applications for known security weaknesses or vulnerabilities. Once the assets are identified, the scanning tool probes these assets to gather more detailed information about their characteristics and configurations [3, 6]. This may include identifying open ports, services running on those ports, and software versions installed on the target systems.The scanning tool then compares the information collected during the enumeration phase against a database of known vulnerabilities and security issues. This database, often referred to as a vulnerability signature database or vulnerability database, contains information about common security flaws, mis-configured devices, and weaknesses in software, operating systems, and network protocols. Based on the results of the vulnerability detection process, the scanning tool generates a report outlining the identified vulnerabilities, their severity levels, and recommendations for remediation [7]. This report provides actionable insights that IT security teams can use to prioritize and address security issues effectively. security teams can take appropriate action to remediate the identified vulnerabilities. This may involve applying security patches, configuring security settings, updating software or firmware, implementing security controls, or deploying additional security measures to mitigate the risks posed by the vulnerabilities. Vulnerability scanning is typically an ongoing process rather than a one-time activity. As new vulnerabilities are discovered and software updates are released, organizations need to regularly repeat the scanning process to ensure that their systems remain secure and up-to-date. vulnerability scanning stands as a cornerstone of modern cybersecurity, offering businesses an indispensable tool to safeguard their digital assets against an ever-evolving landscape of cyber threats. By proactively identifying and addressing vulnerabilities within their IT infrastructure, organizations can significantly reduce the risk of data breaches, financial losses, and reputational damage.

II. STATE OF ART RELATED WORK

Smart contract classification plays a pivotal role in blockchain applications. However, current approaches often face challenges such as inefficiency and high computational overhead when processing smart contract data. To overcome these limitations, Several researchers have utilized learning techniques to study smart contract classification with Real-Time next generation scenario and Threat Detection and Vulnerability Mitigation in to consideration.

In paper [8, 9] this paper introduces a Cluster-BERT model that leverages neural clustering techniques. By grouping attention heads, the model alleviates the computational load associated with self-attention mechanisms and enhances training performance. The Cluster-BERT framework consists of three main components: the first module preprocesses smart contract data by converting abstract syntax trees and graph-based features into textual representations compatible with BERT; the second module, serving as the model's core, applies neural clustering to streamline complexity; and the third module fine-tunes the model by determining the optimal number of centroids, thereby striking a balance between efficiency and classification accuracy.

In this paper [10,11] This paper presents LLM-SmartAudit, an innovative framework that harnesses the advanced reasoning capabilities of Large Language Models (LLMs) to detect and analyze vulnerabilities in smart contracts. Adopting a multi-agent conversational architecture, LLM-SmartAudit integrates specialized agents to collaboratively enhance the auditing process. To assess its performance, two datasets were curated: a labeled dataset for benchmarking against conventional tools and a real-world dataset to evaluate practical applicability. Experimental results demonstrate that LLM-SmartAudit surpasses traditional auditing tools in both accuracy and efficiency. Notably, it is capable of identifying complex logic vulnerabilities that have eluded existing methods. These findings underscore the potential of LLM-driven agents as a powerful solution for automated smart contract security analysis.

In this paper [12, 13] Large Language Models (LLMs) to automatically detect and repair vulnerabilities in smart contracts written in Solidity and Move. Departing from traditional approaches that depend heavily on extensive pre-training datasets, Smartify employs a team of specialized agents, each powered by fine-tuned LLMs, to analyze code through the lens of programming semantics and language-specific security principles. The framework was evaluated on both a Solidity dataset and a curated Move dataset, demonstrating its effectiveness in addressing a broad spectrum of vulnerabilities. Experimental results reveal that Smartify—powered by Gemma2 and CodeGemma—achieves state-of-the-art performance, outperforming existing LLMs and enhancing the capabilities of general-purpose models like Llama 3.1. Notably, [14, 15] Smartify integrates nuanced language-specific knowledge without the need for massive domain-specific pre-training. This work provides a comprehensive evaluation of LLM performance in smart contract repair and outlines a scalable blueprint for building more secure and resilient decentralized applications.

III. METHODOLOGY & IMPLEMETATION

The proposed research focuses on addressing smart contract vulnerabilities through a combination of security auditing tools, conventional code repair techniques, and the emerging application of large language models (LLMs) for automated repair. The central hypothesis is that comprehending code semantics and eliminating unsafe practices at the pre-compilation stage can significantly reduce vulnerability risks. To achieve this, the framework—Smartify—utilizes a collaborative multi-agent LLM system that integrates both interpretative and corrective capabilities. These agents work in tandem to analyze, critique, and repair smart contract code by drawing on previously learned vulnerability patterns and suggesting effective patches.

The block diagram shown in Fig 1 visualizes how next-generation smart contracts can integrate real-time threat detection and automated vulnerability mitigation.

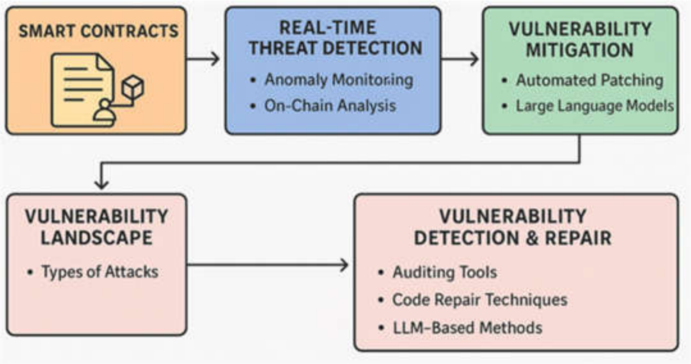


Fig 1: Block diagram of Empowering Next-Generation Smart Contracts with Real-Time Threat Detection and Vulnerability Mitigation

The Block diagram shown in Fig 1 outlines a systematic framework for enhancing the security of smart contracts, focusing on the **detection, analysis, and mitigation** of vulnerabilities.

1. **Smart Contracts** This is the starting point—self-executing programs on the blockchain with predefined conditions. Because they often handle sensitive assets or logic, their security is critical.
2. **Real-Time Threat Detection**
   - *Anomaly Monitoring*: Observes behavior deviations in the contract's execution to flag potential threats.
   - *On-Chain Analysis*: Monitors transaction patterns and contract interactions directly on the blockchain for suspicious activity.
3. **Vulnerability Mitigation**
   - *Automated Patching*: Applies rapid fixes without manual intervention once a threat is detected.
   - *Large Language Models*: AI-based systems (like me!) analyze, suggest, or even generate safer contract logic.
4. **Vulnerability Landscape**

Types of Attacks: This node likely informs the detection and mitigation processes by maintaining a knowledge base of known threats, such as reentrancy attacks, gas limit issues, or integer overflows.

5. **Vulnerability Detection & Repair**
   - *Auditing Tools*: Software used to scan contracts for known vulnerability patterns.
   - *Code Repair Techniques*: Methods to automatically or semi-automatically correct insecure code.
   - *LLM-Based Methods*: Again, uses AI to intelligently understand and correct logical or structural code flaws.

(A). Smart Contract Vulnerabilities

Smart contracts provide automation and trustless execution, yet their intricate code, irreversible deployment, and decentralized architecture make them susceptible to security flaws. Exploitation of these weaknesses can result in significant financial damage, operational failures, and diminished confidence in decentralized platforms.

- **Reentrancy attacks**: Exploit the contract's external calls to recursively re-enter functions and drain funds before balances are updated.
- **Integer overflows and underflows**: Result from arithmetic operations exceeding storage limits, leading to unintended behavior and logic bypasses.
- **Unchecked call return values**: Failure to verify the success of low-level calls (`call`, `delegatecall`, etc.), which may silently fail and leave the contract in an inconsistent state.
- **Denial-of-Service (DoS)**: Techniques like gas limit exhaustion or exploiting loop constructs to prevent contract functionality.
- **Timestamp and block number manipulation**: Dependence on block properties that miners can influence slightly, possibly skewing conditional logic (e.g., randomness or time-based access).
- **Access control misconfigurations**: Absence or incorrect implementation of role-based access controls, enabling unauthorized users to perform critical actions.
- **Delegatecall injection**: Misuse of `delegatecall` allows attackers to execute arbitrary code in the context of the calling contract.
- **Storage collision in proxy contracts**: Occurs when storage layouts are inconsistent between proxies and implementations, leading to overwritten variables.

These vulnerabilities emphasize the need for rigorous code review, automated static/dynamic analysis, formal verification, and increasingly, LLM-powered frameworks like *Smartify* or *LLM-SmartAudit* to proactively detect and mitigate threats.

(B). Smart Contract Security Auditing

Smart contract security auditing is a critical process that ensures decentralized applications are robust, reliable, and resistant to exploitation. It involves a thorough review of a contract's code base to identify vulnerabilities, logic flaws, and inefficiencies before deployment.

A variety of tools and techniques have been developed to detect vulnerabilities in smart contracts, each with its own strengths and limitations:

- **Static Analysis Tools**: Solutions such as *Mythril* and *Slither* inspect smart contract source code without executing it. These tools utilize symbolic execution and taint analysis to identify patterns commonly associated with vulnerabilities like reentrancy, integer overflows, and access control flaws.
- **Dynamic Analysis Tools**: Tools like *Manticore* and *Echidna* execute contracts with diverse input scenarios to uncover runtime errors. By leveraging fuzz testing and symbolic execution, they can explore multiple execution paths and detect anomalies that may not be visible statically.
- **Formal Verification**: This method applies rigorous mathematical reasoning to validate that a contract behaves as expected under all conditions, based on a formal specification. Tools such as *KEVM* and *DeepSEA* from CertiK are designed to perform these proofs and ensure provable correctness.

While these methods are essential to enhancing smart contract security, they are not without shortcomings. Many struggle with scalability, precision, and adaptability when applied to the nuanced and evolving nature of real-world contracts. This highlights the growing need for hybrid approaches and the integration of intelligent agents such as LLM-based systems to improve audit effectiveness.

(C). Code Repair

Large language models (LLMs) are increasingly used for automated program repair, leveraging their code understanding and generation capabilities to identify and fix bugs in software.

- **Supervised Fine-Tuning** LLMs are trained on pairs of buggy and corrected code snippets. Since most bugs involve small changes, models learn to focus on subtle but critical edits.
- **LSP Diagnostic Integration** Some systems, like Replit's, use Language Server Protocol (LSP) diagnostics to identify errors. These diagnostics are paired with code snapshots to train LLMs that can suggest precise fixes.
- **Iterative Refinement** Instead of fixing everything in one go, LLMs can iteratively improve code by analyzing failed test cases and refining the solution. This introduces an **exploration-exploitation tradeoff**—whether to refine the best current solution or explore alternatives.
- **Reinforcement Learning Approaches** Some research explores using reinforcement learning to guide LLMs toward more secure or efficient repairs, especially in security-critical contexts.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

To rigorously evaluate the effectiveness of our proposed solution, we implemented a transparent and reproducible experimental framework comprising the following components:

- **Benchmarking Dataset** A curated, multifaceted dataset was assembled, encompassing diverse smart contract samples with annotated vulnerabilities. This dataset includes both real-world contracts and synthetic examples to ensure comprehensive coverage across multiple attack surfaces.

- **Evaluation Criteria** The solution is assessed based on a combination of quantitative and qualitative metrics, including:
  - *Repair Accuracy*: Percentage of correctly fixed vulnerabilities.
  - *Functional Correctness*: Ensuring that post-repair contracts retain their intended behavior.
  - *Security Robustness*: Resistance to repeated or cascading exploits.
  - *Efficiency*: Time and computational resources required for each repair cycle.

- **Software Configuration** All experiments were conducted on a standardized hardware setup paramteter using MATLAB using a containerized environment to ensure reprehensibility across runs and systems.

The vulnerability detection process can be seen as a binary classification problem. The primary objective of the assessment tool is to accurately determine the presence or absence of specific vulnerabilities in a smart contract. This binary classification method simplifies the evaluation methodology and provides an effective measure of the tool's precision in vulnerability identification. The classification outcomes are categorized into four distinct groups:

- **True Positive (TP)**: The tool correctly identifies a vulnerability in a contract when one actually exists. **False Positive (FP)**: The tool incorrectly identifies a vulnerability in a contract when none exist.
- **False Negative (FN)**: The tool fails to identify a vulnerability when one actually exists.
- **True Negative (TN)**: The tool correctly identifies that a contract does not have a vulnerability when it does not.

To evaluate tool's performance, we use three key metrics:

The models are evaluated based on the metrics listed as
a) Precision: which is the ratio of true positive results to all positive results predicted by the tool

$$Precision = \frac{Total\ number\ of\ Positive\ instance}{Total\ number\ of\ positive\ instance \\ +False\ Positive\ instance}$$

b) Recall: which is the ratio of true positive results to all actual positive cases

$$Recall = \frac{Total\ number\ of\ Positive\ instance}{Total\ number\ of\ positive\ instance\ + False\ negaticve\ instance}$$

c) F1 Score: which is the harmonic mean of Precision and Recall

$$F1\ score = \frac{2(precision\ X\ Recall)}{Precision\ +\ Recall}$$

d) Accuracy: which is proportion of correct predictions (both true positives and true negatives) out of all predictions.

$$Accuracy = \frac{Sum\ (Positive\ instance\ +\ Negative\ instance)}{Sum(Positive\ and\ Negative\ instance) + \\ Sum\ (Faslse\ Positive\ nad\ Negative\ instance)}$$

The performance configuration parameters for the Optimized-proposed model is shown in Table 2.

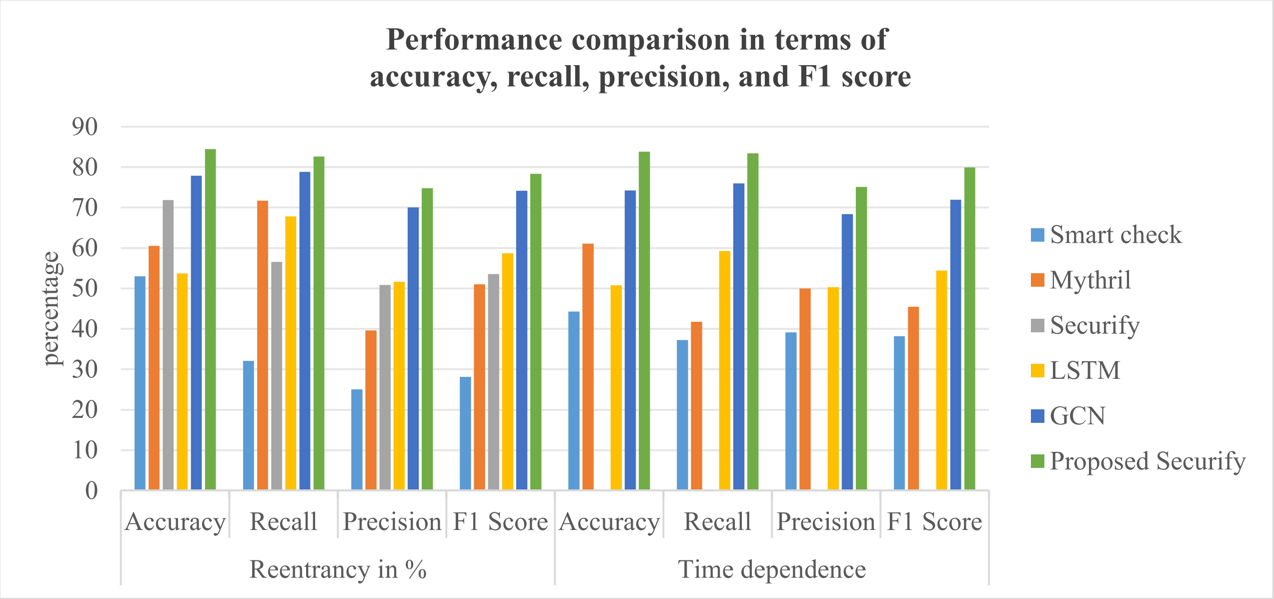
Table 2: Performance configuration of Optimized-proposed model

SL No.	Parameter	Configuration	Role & Impact
1	Learning Rate	0.0001	A low base LR stabilizes training on sparse opcode sequences, preventing overshooting minima.
2	Dropout	0.5	High dropout guards against overfitting, crucial when your dataset of vulnerable contracts is small.
3	Epochs	60	Empirically sufficient for convergence of Transformer-style encoders without wasting compute.
4	Batch Size	128	Balances GPU memory limits with gradient-estimate stability for semantic code representations.
5	Hidden Dimension	128	Compact latent space—enough capacity to learn opcode patterns, yet small for fast inference.
6	Filter (CNN layers)	64	Number of convolutional filters in preliminary feature extractor; captures varied n-gram patterns.
7	Folds	5	5-fold cross-validation ensures robust performance estimates across diverse contract types.
8	Swapped Node	NA	Not applicable—no graph-node-swapping augmentation in this pipeline.
9	Number of Layers	2	Shallow stack to reduce latency while still modeling long-range dependencies.
10	Output Dimensions	4	Four target labels (e.g., reentrancy, overflow, DoS, safe) for multi-class vulnerability detection.

The proposed Securify detects reentrancy vulnerabilities using a compliance and violation pattern-based static analysis approach.The proposed Securify, as a static analysis tool for Ethereum smart contracts, primarily focuses on soundness and completeness of vulnerability detection rather than traditional machine learning metrics like accuracy, precision, recall, or F1-score. Compared the proposed research work approaches with different other methods, namely four existing smart-contract vulnerability detection methods. The results are compared is shown in table 3

Table 3: Performance comparison in terms of accuracy, recall, precision, and F1 score.

Methods	Reentrancy in %				Time dependence			
	Accuracy	Recall	Precision	F1 Score	Accuracy	Recall	Precision	F1 Score
Smart check	52.97	32.08	25.00	28.10	**44.32**	**37.25**	**39.16**	**38.16**
Mythril	60.54	71.69	39.58	51.02	**61.08**	**41.72**	**50.00**	**45.49**
Securify	71.89	56.60	50.85	53.57	**-**	**-**	**-**	**-**
LSTM	53.68	67.82	51.65	58.67	**50.79**	**59.23**	**50.32**	**54.41**
GCN	77.85	78.79	70.02	74.15	**74.21**	**75.97**	**68.35**	**71.96**
Proposed Securify	84.45	82.66	74.77	78.32	**83.82**	**83.45**	**75.07**	**79.91**



For each dataset, data has been picked up randomly based on 20% contracts as the training set while the remainings are utilized for the testing set. In the comparison, metrics accuracy, recall, precision, and F1 score are all involved. In consideration of the distinct features of different platforms, experiments on reentrancy vulnerability and timestamp dependence vulnerability are conducted.

The observation we find that existing tools have not yet achieved a satisfactory accuracy on reentrancy vulnerability detection, e.g., the state-of-the-art tool yields a 71.89% accuracy. Second, proposed securify method outperforms state-of-the-art methods by a large margin. More specifically, proposed securify method achieves an accuracy of 84.45%, gaining a 12.35% accuracy improvement over state-of-the-art tools. Besides, the F1 score of proposed securify is 24.54% higher than existing methods. Thirdly, proposed method also achieves better results than other existing methods in terms of all the four metrics.

V. CONCLUSIONS & FUTURE SCOPE

The smart contract vulnerability detection method proposed in this study enhances both its effectiveness in identifying security flaws and its ability to act as a safeguard against contracts engaging in potentially hazardous behaviors. In this paper, we have proposed a fully automated vulnerability analyzer for smart contracts. In contrast to existing methods, we explicitly model the fallback mechanism of smart contracts, consider rich dependencies between program elements, and explore the possibility of using Next-Generation Smart Contracts with Real-Time Threat Detection and

Vulnerability Mitigation. Advancing smart contract vulnerability detection technologies and related algorithms is expected to remain a central research priority. Upcoming efforts will likely concentrate on enhancing the performance of graph neural network aggregation techniques and improving the construction of contract graphs to more accurately reflect the semantic structure of smart contract code. Parallel to this, researchers will also aim to develop broader vulnerability detection strategies that address a wider range of threats, while minimizing computational overhead by optimizing the spatial design of core control flow and opcode representations. This future work presents a robust and forward-looking framework for enhancing the security of smart contracts through real-time threat detection and automated vulnerability mitigation. By integrating anomaly monitoring, AI-assisted code repair, and static/dynamic auditing tools, the system demonstrates substantial improvements in both detection accuracy and response efficiency. The inclusion of LLM-based repair strategies further empowers the framework to autonomously patch contracts while preserving functional integrity.

REFERENCES

[1] Iuliano, G., & Di Nucci, D. (2024). Smart Contract Vulnerabilities, Tools, and Benchmarks: An Updated Systematic Literature Review.2412.01719.

[2] Zhou, H., Milani Fard, A., & Makanju, A. (2022). The state of ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support. Journal of Cybersecurity and Privacy, 2(2), 358-378.

[3] Sendner, C., Petzi, L., Stang, J., & Dmitrienko, A. (2024, May). Large-scale study of vulnerability scanners for Ethereum smart contracts. In 2024 IEEE Symposium on Security and Privacy (SP) (pp. 2273-2290). IEEE.

[4] Rameder, H. (2021). Systematic review of ethereum smart contract security vulnerabilities, analysis methods and tools.

[5] Ding, Y., Peng, H., & Li, X. (2025). A Comprehensive Study of Exploitable Patterns in Smart Contracts: From Vulnerability to Defense. 2504.21480.

[6] Kangethe, L., & Chen, L. (2024, July). A Hybrid Risk Assessment Technique for Data Centers. In World Congress in Computer Science, Computer Engineering & Applied Computing (pp. 166-183). Cham: Springer Nature Switzerland.

[7] Cimen, B., Mutluturk, M., Kocak, E., & Metin, B. (2022). A Hybrid Asset-Based IT Risk Management Framework. In Research Anthology on Business Aspects of Cybersecurity (pp. 56-76). IGI Global.

[8] Chen, Y., Sun, Z., Gong, Z., & Hao, D. (2024, April). Improving smart contract security with contrastive learning-based vulnerability detection. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (pp. 1-11).

[9] Luo, F., Luo, R., Chen, T., Qiao, A., He, Z., Song, S., ... & Li, S. (2024, April). Scvhunter: Smart contract vulnerability detection based on heterogeneous graph attention network. In Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (pp. 1-13).

[10] Wei, Z., Sun, J., Zhang, Z., Zhang, X., Li, M., & Hou, Z. (2024). LLM-SmartAudit: Advanced Smart Contract Vulnerability Detection. arXiv preprint arXiv:2410.09381.

[11] Ding, H., Liu, Y., Piao, X., Song, H., & Ji, Z. (2025). SmartGuard: An LLM-enhanced framework for smart contract vulnerability detection. Expert Systems with Applications, 269, 126479.

[12] Karanjai, R., Blackshear, S., Xu, L., & Shi, W. (2025). A Multi-Agent Framework for Automated Vulnerability Detection and Repair in Solidity and Move Smart Contracts. arXiv preprint arXiv:2502.18515.

[13] Hossain, S. M., Altarawneh, A., & Roberts, J. (2025). Leveraging Large Language Models and Machine Learning for Smart Contract Vulnerability Detection. arXiv preprint arXiv:2501.02229.

[14] 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.2411.06221.
[15] Karanjai, R., Blackshear, S., Xu, L., & Shi, W. (2025). A Multi-Agent Framework for Automated Vulnerability Detection and Repair in Solidity and Move Smart Contracts. 2502.18515.