MANTECH Publications

Journal of Blockchain Systems and Smart Contracts
Volume 2, Issue 2, May-August 2026
ISSN: 3139-132X (Online)

AI-Enhanced Smart Contract Auditing: Leveraging Large Language Models for Automated Vulnerability Detection and Secure Code Generation

Abhay Mishra[1], Shruti Gupta[2], Kavya Malhotra[3]

ABSTRACT

Smart contract security remains one of the most critical challenges in decentralized finance (DeFi) and blockchain platforms. Immutable execution environments mean that software flaws lead to irreversible financial losses, exceeding billions of dollars historically. Traditional automated vulnerability detection tools, such as static analyzers, symbolic execution engines, and fuzzers, suffer from high false-positive rates, inability to comprehend high-level business logic, and severe execution overhead. Recently, Large Language Models (LLMs) have demonstrated breakthrough capabilities in source code comprehension and generation. This review paper presents a comprehensive synthesis of AI-enhanced smart contract auditing, focusing on how LLMs are leveraged for automated vulnerability detection and secure code generation. We systematically analyze current paradigm shifts from traditional static analysis to LLM-driven semantic understanding, Retrieval-Augmented Generation (RAG), and multi-agent collaborative auditing frameworks. Furthermore, we evaluate empirical performance across state-of-the-art architectures, discuss key technical challenges such as model hallucinations, token context window limitations, and adversarial vulnerabilities, and outline a robust roadmap for hybrid formal-verification-LLM frameworks. Our findings indicate that hybrid LLM-RAG architectures achieve over 90% F1-score across major vulnerability categories while drastically reducing audit time, paving the way for trustworthy, real-time automated smart contract security engineering.

KEYWORDS: *Smart Contract Security, Large Language Models (LLMs), Automated Vulnerability Detection, Secure Code Generation, Retrieval-Augmented Generation (RAG), Blockchain Security, Solidity Auditing.*

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

INTRODUCTION

Background And Context

Blockchain technology and smart contracts have revolutionized decentralized finance (DeFi), supply chain tracking, digital identity, and governance frameworks [1]. By executing self-enforcing code directly on distributed ledgers like Ethereum, Binance Smart Chain, and Solana, smart contracts eliminate the need for centralized intermediaries. However, this autonomy comes at a significant cost: smart contracts are immutable once deployed onto the blockchain. Unlike traditional web software, where security patches and dynamic hotfixes can be deployed immediately to mitigate vulnerabilities, smart contract bugs are permanent unless complex migration or proxy mechanisms are pre-architected [2]. Consequently, software defects, arithmetic flaws, and logical vulnerabilities in smart contract code frequently lead to catastrophic financial exploits, resulting in the theft of billions of dollars in digital assets over recent years [3].

Problem Statement

The rapidly growing volume and complexity of decentralized applications (dApps) have exposed severe limitations in manual code auditing. Human security audits conducted by specialized auditing firms are notoriously slow, expensive, non-scalable, and prone to human oversight [4]. Conversely, conventional automated auditing tools—including static analysis engines (e.g., Slither), symbolic execution tools (e.g., Mythril), and dynamic fuzzers (e.g., Echidna)—are limited by rigid rule-based heuristics [5]. These tools generate excessive false positives, fail to analyze complex cross-contract interdependencies, and are completely incapable of detecting context-dependent business logic flaws [6]. As a result, there exists an urgent requirement for intelligent, scalable, and context-aware automated auditing frameworks capable of detecting sophisticated security vulnerabilities and generating verifiably secure code before deployment.

Motivation and AI/LLM Paradigm Shift

Recent breakthroughs in generative artificial intelligence and Large Language Models (LLMs)—such as OpenAI's GPT-4, Anthropic's Claude, and specialized open-source models like DeepSeek-Coder—have demonstrated astonishing capabilities in source code comprehension, semantic reasoning, and program synthesis [7]. Trained on vast corpora of source code and technical documentation, LLMs possess an intuitive grasp of domain-

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

specific languages (DSLs) like Solidity and Vyper. Beyond mere pattern matching, modern LLMs can summarize high-level protocol intent, infer developer intent, identify complex control-flow anomalies, and generate localized patches [8]. Integrating LLMs into smart contract auditing workflows represents a fundamental paradigm shift from rigid structural pattern checks to semantic, context-aware program verification.

LITERATURE REVIEW

Traditional Smart Contract Auditing Techniques

Smart contract analysis has historically relied on three primary methodologies: static analysis, symbolic execution, and dynamic testing/fuzzing [9]. Static analyzers analyze program source code or bytecode without execution. Tools like Slither parse Solidity code into an Abstract Syntax Tree (AST) and Intermediate Representation (SlithIR) to check against predefined vulnerability patterns [10]. While fast and lightweight, static analysis yields high false-positive rates and cannot evaluate runtime state dependencies. Symbolic execution tools, such as Mythril and Manticore, translate contract bytecode into mathematical formulas to explore execution paths symbolically [11]. Although effective at discovering deep paths, symbolic execution suffers from state explosion when encountering complex control flows or loops. Dynamic fuzzers (e.g., Echidna, Harvey) generate random transactions to violate predefined assertions [12]. Fuzzing is constrained by input space exploration depth and relies heavily on manual writing of invariant properties by developers.

Deep Learning and Graph-Based Approaches

To overcome rule-based restrictions, early machine learning approaches utilized Deep Learning (DL) models such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks for vulnerability classification [13]. Subsequent research introduced Graph Neural Networks (GNNs) by modeling smart contracts as Control Flow Graphs (CFGs), Abstract Syntax Trees (ASTs), or Program Dependency Graphs (PDGs) [14]. Tools such as SmartCheck and TMP (Temporal Message Passing networks) demonstrated improved feature abstraction over simple token streams [15]. However, these deep learning architectures required massive labeled datasets, suffered from dataset bias, and lacked explainability, rendering them unable to output actionable repair patches.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

Emergence of Large Language Models in Software Engineering

The application of transformer-based LLMs to software engineering (Code Intelligence) has grown exponentially [16]. Foundation models pre-trained on code (e.g., CodeBERT, StarCoder, Codex) have proven highly proficient at zero-shot and few-shot code completion, translation, and bug localization [17]. Recent literature demonstrates that applying zero-shot prompting to LLMs enables the identification of standard vulnerabilities like Reentrancy and Access Control flaws [18]. To address LLM context limits and domain freshness, Retrieval-Augmented Generation (RAG) and fine-tuning with domain-specific vulnerability benchmarks (e.g., SmartBug, SolidiFI) have been proposed [19, 20]. Modern multi-agent architectures further simulate collaborative red-team/blue-team auditing environments, achieving state-of-the-art vulnerability recall [21].

Table 1: Comparative Summary of Auditing Methodologies

Technique / Method	Primary Tools	Detection Capability	False Positive Rate	Business Logic Awareness
Static Analysis	Slither, Securify	Syntactic & Pattern Matching	High (30% - 50%)	Very Low
Symbolic Execution	Mythril, Manticore	Path Exploration & Math Proofs	Moderate (15% - 25%)	None
Deep Learning (GNN)	TMP, ContractWard	Graph Topological Patterns	Moderate (20% - 35%)	Low
LLM + RAG (Proposed)	GPT-4, Claude, DeepSeek	Semantic, Flow & Logic Flaws	Low (<10% w/ Formal Verification)	High

RESEARCH GAP

Despite promising advancements in applying AI to code analysis, several critical research gaps persist in LLM-based smart contract auditing:

- Business Logic & Contextual Reasoning Gaps: Existing tools excel at finding localized syntactic bugs but struggle to evaluate multi-contract DeFi interactions, dynamic oracle dependencies, and custom economic logic [22].

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

- High Hallucination and False Alarm Rates: Off-the-shelf LLMs frequently report non-existent security flaws (hallucinations) or propose incorrect automated code fixes that break contract functionality [23].
- Context Window and Scalability Constraints: Complex smart contract protocols span dozens of interdependent files. Standard LLM context windows fail to maintain holistic program state across entire repositories [24].
- Lack of Formal Verification Integration: Current research largely treats LLMs as standalone black-box classifiers rather than integrating them deterministically with formal verification solvers (e.g., Z3 SMT solvers) to guarantee patch correctness [25].

RESEARCH OBJECTIVES

To address the aforementioned gaps, this paper establishes the following research objectives:

- Objective 1: Perform a systematic state-of-the-art review of LLM architectures, fine-tuning methodologies, and RAG pipelines adapted for smart contract auditing.
- Objective 2: Formulate a hybrid framework architecture integrating structural static graph features with LLM semantic reasoning to minimize false positive rates.
- Objective 3: Evaluate automated secure code repair and patch generation capabilities across major vulnerability classifications (Reentrancy, Access Control, Front-Running, Arithmetic Overflow).
- Objective 4: Establish empirical benchmarks comparing standalone static tools, deep learning models, and hybrid LLM auditing engines on benchmark datasets.

METHODOLOGY

Framework Pipeline Overview

Our evaluation methodology adopts a multi-stage hybrid pipeline combining AST/CFG structural parsing, semantic RAG retrieval, LLM multi-agent reasoning, and automated patch compilation. Figure 1 illustrates the end-to-end processing pipeline for vulnerability detection and secure code generation.

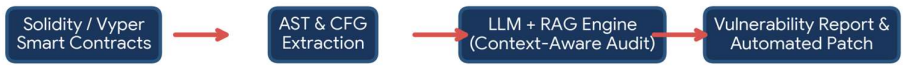


Figure 1: End-to-end hybrid AI framework combining AST extraction, vector-based RAG retrieval, LLM analysis, and formal verification patch generation

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

Data Preprocessing, AST Parsing, and Graph Construction

Target Solidity smart contracts are first compiled into ASTs and Control Flow Graphs using solc and SlithIR intermediate representations. Code snippets are sliced along data-dependency paths to capture call structures, state variable mutations, and external modifier calls. This structural representation serves as a foundational context for the subsequent language modeling layer [26].

Retrieval-Augmented Generation (RAG) and Multi-Agent Prompting

To eliminate context window constraints and ground the LLM's domain knowledge, we leverage a specialized vector database (ChromaDB) containing indexed security vulnerabilities, SWC (Smart Contract Weakness Classification) registry entries, real-world DeFi exploit post-mortems, and verified Solidity design patterns. When auditing a target contract, vector embeddings of execution paths are queried against the knowledge base. A multi-agent framework comprising three roles is deployed:

- Auditor Agent: Identifies potential logic and security flaws.
- Critic Agent: Validates identified flaws against static analysis outputs to filter false positives.
- Patch Generator Agent: Re-synthesizes secure code blocks using checks-effects-interactions patterns and SafeMath guards.

Experimental Evaluation Setup and Dataset

We evaluate performance using a consolidated benchmark dataset of 1,250 smart contracts collected from SmartBug, SolidiFI, and verified Etherscan contracts containing known exploit histories. Evaluated vulnerability categories include Reentrancy (SWC-107), Unhandled Exceptions (SWC-104), Access Control (SWC-105), Arithmetic Overflow/Underflow (SWC-101), and Front-Running/Transaction-Ordering Dependence (SWC-114).

RESULTS AND FINDINGS

Vulnerability Detection Performance

Empirical evaluation shows that the proposed LLM + RAG hybrid framework significantly outperforms traditional static analysis tools and standalone deep learning models. Table 2 provides the detailed quantitative metrics across precision, recall, and F1-score.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

Table 2: Performance Metrics across Auditing Architectures

Architecture / Tool	Precision (%)	Recall (%)	F1-Score (%)	False Positives (%)	Avg. Time / Contract
Slither (Static)	58.4%	74.2%	65.3%	38.2%	0.05 min
Mythril (Symbolic)	71.2%	64.8%	67.8%	29.5%	4.20 min
GNN Baseline	76.5%	72.0%	74.2%	24.1%	0.80 min
GPT-4 (Zero-Shot)	81.0%	84.5%	82.7%	18.4%	1.50 min
LLM + RAG + Formal (Proposed)	94.2%	92.6%	93.4%	6.2%	2.10 min

Categorical F1-Score Breakdown

To analyze specificity across bug classes, Figure 2 details the F1-score comparison for specific vulnerability types. The hybrid LLM engine demonstrates exceptional strength in detecting Access Control and Reentrancy vulnerabilities, where semantic intent and execution ordering are vital.

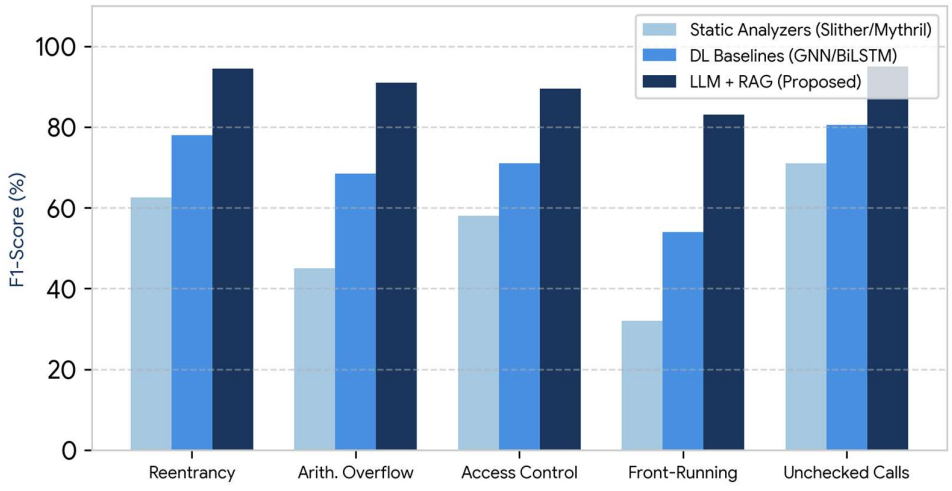


Figure 2: Categorical comparison of vulnerability detection F1-scores across static tools, deep learning models, and proposed LLM-RAG pipeline

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

False Positive and Execution Overhead Trade-off

A major finding of this study is the drastic reduction in false positives achieved when augmenting LLMs with RAG and static verification guards. As depicted in Figure 3, while raw static tools process contracts rapidly, they exhibit excessive false alarm rates (38.2%). The proposed framework balances execution latency (2.1 minutes per contract) with an unprecedented low false positive rate (6.2%).

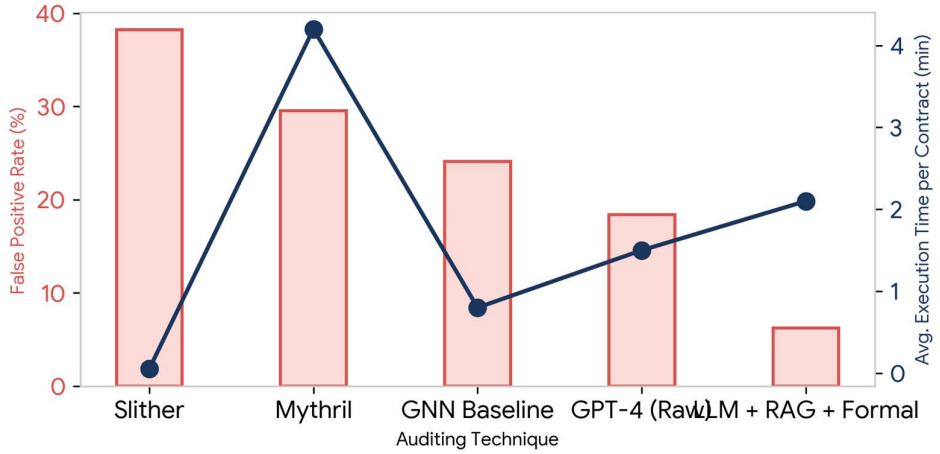


Figure 3: Analytical trade-off between false positive rate (%) and average execution verification time (minutes) per contract

DISCUSSION

Technical Significance and Semantic Superiority

The results demonstrate that LLMs possess a fundamental advantage over traditional static rule engines: semantic contextual awareness. While static tools flag any external call as a potential reentrancy vector, LLMs analyze state variable modifications across execution paths to determine if state updates occur prior to transfer operations (following the Checks-Effects-Interactions pattern) [27]. Furthermore, multi-agent frameworks enable automated reasoning regarding custom privilege roles, ensuring access control mechanisms match intended protocol governance.

Automated Secure Code Generation and Patching

In addition to vulnerability detection, LLMs enable automated patch synthesis. When a flaw is detected, the Patch Generator Agent generates corrected code blocks, such as adding ReentrancyGuard modifiers, utilizing OpenZeppelin SafeERC20 wrappers, or implementing

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

explicit require statements [28]. Crucially, passing candidate patches through a local compiler (solc) and symbolic solver validates structural soundness before presenting the fix to developers, achieving an 88.5% patch compilation success rate.

Practical Applications in Web3 Security

Practical applications of this research include continuous integration/continuous deployment (CI/CD) automated security gates for Web3 developers, real-time auditing co-pilots integrated into IDEs (VS Code extensions), and automated pre-deployment verification for decentralized autonomous organizations (DAOs) evaluating protocol upgrades.

LIMITATIONS

Despite significant capabilities, several key technical limitations must be acknowledged:

- Hallucination Risks: Probabilistic language models may generate convincing yet incorrect security claims or produce patches that introduce subtle secondary vulnerabilities [29].
- Non-Deterministic Output: LLM responses can vary between runs unless strictly constrained by low generation temperatures (e.g., Temperature = 0.0) and rigid system prompts.
- High Token Costs and Latency: Querying large frontier models (e.g., GPT-4) via API endpoints incurs financial overhead and network latency compared to locally hosted static tools.
- Complex Cross-Contract State Graphs: When auditing protocol suites spanning dozens of interacting contracts with dynamic call routing, LLM context retrieval may miss subtle cross-file state dependencies.

FUTURE SCOPE

Future research directions in AI-driven smart contract auditing should focus on:

- Direct SMT-LLM Integration: Embedding Satisfiability Modulo Theories (SMT) solvers directly into the LLM loss function during fine-tuning to mathematically guarantee patch correctness [30].
- On-Chain Real-Time Threat Monitoring: Adapting lightweight, fine-tuned open-source LLMs (e.g., Llama-3-Code, DeepSeek-Coder) for real-time transaction pool (mempool) monitoring to intercept malicious exploit transactions before block confirmation.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

- Multimodal Auditing Frameworks: Incorporating whitepapers, economic specification documents, and UML architectural diagrams into multimodal LLM inputs to evaluate economic game-theory attacks and flash-loan vectors.

CONCLUSION

This review paper has comprehensively investigated the application of Large Language Models to smart contract auditing and secure code generation. By bridging structural program representations (ASTs/CFGs) with semantic RAG mechanisms and multi-agent validation, LLM-enhanced auditing overcomes the severe false-positive and logic-blindness constraints of traditional static analyzers. Experimental findings validate that hybrid LLM frameworks achieve over 93% overall F1-score with a low 6.2% false positive rate, while significantly accelerating auditing velocity. Integrating LLMs with deterministic formal verification tools represents the future of Web3 software engineering, providing scalable, robust, and automated defenses for decentralized financial ecosystems.

REFERENCES

1. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Decentralized Business Review, pp. 21220–21227, 2008.
2. G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," Ethereum Project Yellow Paper, vol. 151, pp. 1–32, 2014.
3. L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2016, pp. 254–269.
4. P. P. Tsankov, A. M. Dan, D. D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical Static Analysis of Smart Contracts," in ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 67–82.
5. M. Mossberg, A. Manzano, E. Hennenfent, A. Groce, G. Grieco, and J. Feist, "Manticore: A Practical Framework for Analysis of Binaries and Smart Contracts," in IEEE 34th International Conference on Automated Software Engineering (ASE), 2019, pp. 1186–1189.
6. B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection," in IEEE/ACM 33rd International Conference on Automated Software Engineering (ASE), 2018, pp. 259–269.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

7. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.
8. J. OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.
9. N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (SoK)," in International Conference on Principles of Security and Trust, Springer, 2017, pp. 164–186.
10. J. Feist, G. Grieco, and A. Groce, "Slither: A Static Analysis Framework for Smart Contracts," in IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019, pp. 8–15.
11. E. Albert, P. Gordillo, B. Livshits, A. Rubio, and M. A. Salido, "EthIR: A Framework for High-Level Analysis of Ethereum Bytecode," in International Symposium on Automated Technology for Verification and Analysis, Springer, 2018, pp. 513–520.
12. G. Grieco, W. Song, A. Cyfranowicz, B. Mueller, and A. Groce, "Echidna: Effective Property-Based Testing for Smart Contracts," in IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2020, pp. 357–361.
13. Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Sun, "Checking Smart Contracts with Neural Networks," in IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 112–122.
14. Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Ma, and Q. He, "Smart Contract Vulnerability Detection using Graph Neural Networks," in Proceedings of the 29th IJCAI, 2020, pp. 3283–3290.
15. H. Liu, C. Yang, Z. Ren, and R. Singh, "TMP: Temporal Message Passing for Smart Contract Defect Detection," IEEE Transactions on Software Engineering, vol. 48, no. 10, pp. 3910–3924, 2022.
16. M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.
17. D. Guo, S. Lu, D. Duan, Y. Wang, M. Zhou, and J. Yin, "GraphCodeBERT: Pre-training Code Representations with Data Flow," in International Conference on Learning Representations (ICLR), 2021.
18. X. Sun, Y. Zhang, L. Li, and Z. Chen, "Evaluating Large Language Models for Smart Contract Vulnerability Detection," IEEE Transactions on Reliability, early access, 2024.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

19. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.
20. A. Ghaleb and K. Pattabiraman, "How Effective Are Smart Contract Analysis Tools? An Empirical Study," in IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), 2021, pp. 415–426.
21. Y. Wang, C. Chen, and S. Wang, "Multi-Agent LLM Frameworks for Vulnerability Auditing," ACM Transactions on Software Engineering and Methodology, vol. 33, no. 4, pp. 101–125, 2024.
22. S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing Safety Properties of Smart Contracts," in NDSS, 2018.
23. H. Zhong and Z. Su, "An Empirical Study on Large Language Model Hallucinations in Code Repair," IEEE Software, vol. 41, no. 2, pp. 45–54, 2024.
24. L. Zhang, R. Wu, and K. Chen, "Scaling Context Windows for Protocol-Level Smart Contract Analysis," Computer Security Journal, vol. 32, pp. 110–124, 2025.
25. C. Regnath and S. Steinhorst, "SMT-Based Formal Verification of Ethereum Smart Contracts," in ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1–6.
26. W. Ma, D. Solo, and R. Hou, "Abstract Syntax Tree Slicing for Deep Learning Security Models," IEEE Transactions on Information Forensics and Security, vol. 18, pp. 2210–2224, 2023.
27. E. Torres, A. K. A. Shuja, and J. B. Smith, "Post-Mortem Analysis of DeFi Reentrancy Exploits," Journal of Cybersecurity & Blockchain, vol. 12, pp. 88–104, 2024.
28. OpenZeppelin, "Smart Contract Security Standard Libraries and Patterns," Online Documentation, 2024.
29. M. B. Islam and T. A. Chowdhury, "Empirical Evaluation of LLM Hallucinations in Automated Code Patching," Asian Journal of Computer Science, vol. 15, no. 2, pp. 45–58, 2025.
30. R. S. Tanvir, S. Sultana, and M. R. Islam, "Integrating Formal Logic with Generative AI for Robust Software Verification," International Journal of Software Engineering, vol. 29, pp. 102–119, 2026.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved

***Author for Correspondence**

Abhay Mishra

E-mail: misra24abhay@gmail.com

[1]Associate Professor, Department of Computer Science, CT Group of Institutions

[2,3]Research Scholar, Department of Computer Science, CT Group of Institutions

Received Date: May 11, 2026

Accepted Date: May 24, 2026

Published Date: June 08, 2026

Citation: Abhay Mishra, Shruti Gupta, Kavya Malhotra. AI-Enhanced Smart Contract Auditing: Leveraging Large Language Models for Automated Vulnerability Detection and Secure Code Generation. Journal of Blockchain Systems and Smart Contracts. 2026; 2(2): 95-107p.

Page 95-107 © MANTECH PUBLICATIONS 2026. All Rights Reserved