International Journal of Multidisciplinary Research and Publications
ISSN (Online): 2581-6187

A Review of AI-Driven Approaches for Vulnerability Detection in Smart Contract Bytecode

Kun Cheng[1], Shaobo Zhang[1], Sun Zhang[1*]
[1]Anhui University of Finance and Economics, Bengbu City, China-233030
Email address: 3202510240@aufe.edu.cn

***Abstract*—*Smart contracts serve as the cornerstone of trust within the blockchain ecosystem, with their security directly determining the stability of the entire system. However, the swift advancement of decentralized applications has precipitated the emergence of numerous vulnerabilities, thereby intensifying the competition between efficient detection technologies and attack methods. Consequently, contract security has become a focal point of concern in both academic and industrial circles. This paper systematically reviews the literature on smart contract bytecode vulnerability detection from 2020 to 2026, focusing on mainstream detection paradigms driven by deep learning, including time series analysis, Graph Neural Networks (GNNs), Convolutional Neural Networks (CNNs), and large-scale model pre-training techniques. Empirical evidence demonstrates that, compared to traditional static analysis tools, these deep learning models significantly enhance the recall rate and F1 score for common vulnerabilities on benchmark datasets such as SmartBugs Wild and ContractBench. Notably, deep semantic feature extraction exhibits distinct advantages in addressing highly complex logical vulnerabilities. Despite surpassing traditional audit methods in terms of detection coverage, the practical effectiveness of deep learning models remains constrained by several factors: uneven data distribution undermines model robustness, the diversity of vulnerability types leads to a high false positive rate, and existing models typically exhibit a strong reliance on labeled data, a limitation inherent in the supervised learning paradigm. This reliance impedes effective generalization and detection when confronted with zero-day vulnerabilities or novel attack vectors. Furthermore, semantic ambiguities during bytecode decomplication, challenges in adapting to cross-virtual machine environments, the opaque nature of model decision-making, and fragmented evaluation criteria collectively constitute significant barriers to transitioning this technology from theoretical development to engineering application, thereby substantially affecting method reproducibility. In light of these considerations, this study aims to systematically trace the evolution of artificial intelligence-driven bytecode vulnerability detection technologies, assess the technical principles, applicability, and practical performance of each model, and provide a theoretical foundation and conceptual framework for constructing a highly reliable and adaptive smart contract security system.*

Keywords*—*Artificial intelligence; Bytecode; Vulnerability detection; Smart contract.

I. INTRODUCTION

Blockchain technology is undergoing a radical transformation, fundamentally reshaping the foundational architecture of digital trust [1]. As autonomous entities tasked with executing preset protocol logic within this framework, smart contracts [2] are crucial for maintaining order and stability in decentralized ecosystems, owing to the stability of their code. Unlike traditional software systems, smart contracts become immutable once deployed, imposing stringent demands on their fault tolerance. Under such demands, even minor logical flaws can be amplified into systemic risks, triggering a cascade of asset losses and a collapse of trust. With the exponential surge in Web3.0 applications, attackers and defenders have entered a phase of dynamic interaction characterized by rapid iteration: attack paths are progressively becoming more concealed and complex, evolving from simple reentrancy vulnerabilities to multi-dimensional, cross-contract composite attack patterns [3]. In contrast, traditional defence mechanisms, which predominantly rely on rule-based matching or manual audits, have demonstrated significant performance limitations when confronted with vast code repositories and rapidly evolving attack techniques, struggling to balance detection efficiency with comprehensive coverage. Therefore, academia and industry urgently need to explore technical solutions with enhanced generalization capabilities and adaptability [4] to address the escalating technological contest between advanced detection methods and automated attack technologies.

The incorporation of artificial intelligence technology, particularly the advent of deep learning techniques [5], has established an innovative methodological framework for smart contract vulnerability detection [6]. Through an in-depth analysis of the semantic features and spatio-temporal structural information embedded within bytecode, this approach endeavours to surmount the inherent limitations of rule-dependent matching found in conventional static analysis tools [7], thereby enabling the effective identification and extensive detection of previously unknown vulnerability patterns. A retrospective examination of the evolutionary trajectory of blockchain technology [8] since its inception in the early 21st century, with a particular focus on the research advancements over the past decade, reveals that deep learning-based smart contract security analysis has progressively evolved into a multidimensional and comprehensive technological domain.

Currently, solutions based on artificial intelligence continue to confront multiple challenges. For instance, publicly accessible datasets (e.g., SmartBugs Wild [9] and ContractBench) commonly exhibit significant issues such as class imbalance, annotation noise, and limited cross-platform versatility, all of which directly undermine the model's robustness. Furthermore, the high false positive rate further constrains the practical applicability of these tools in industrial settings [10]. Critically, model training heavily relies on historical vulnerability samples, leading to a marked

Kun Cheng and Sun Zhang, "A Review of AI-Driven Approaches for Vulnerability Detection in Smart Contract Bytecode," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 8, Issue 10, pp. 47-52, 2026.

deficiency in detecting novel attack patterns or logical variants. Additionally, the semantic ambiguity [11] arising during bytecode decompilation, insufficient abstraction of contract semantics, and inconsistent evaluation criteria collectively impede the reproducibility and cross-method comparability of the proposed approaches.

This study systematically reviews research literature in the field of smart contract bytecode security detection from 2020 to 2026, with a particular focus on the evolutionary trends in AI applications. The research encompasses the classification framework of mainstream deep learning methods, the underlying technological implementation principles, experimental evaluation outcomes, and their respective application scenarios. Simultaneously, it summarizes the core bottlenecks in data construction, model generalization, and practical deployment observed in current research, providing a clear problem landscape and methodological reference for subsequent studies.

II. Background

In recent years, frequent incidents involving substantial financial losses caused by contract vulnerabilities **Error! Reference source not found.**[12][13] have underscored the urgent need for efficient and accurate automated detection mechanisms. To this end, it is necessary to first clarify the essential structure of the technology stack underpinning smart contract operation, identify the root causes and establish a classification system for common vulnerabilities, and develop a standardized evaluation framework. The following section will systematically review the core background knowledge upon which research in this domain is built.

A. Blockchain Fundamentals

Blockchain is a distributed, decentralized shared ledger technology[15], whose core structure consists of blocks linked chronologically. Each block contains a dataset of transactions, a timestamp, consensus credentials such as proof of work (or proof of stake), and the cryptographic hash of the preceding block, forming a chain-like tamper-resistant structure. Network nodes collaboratively validate transactions and achieve state consistency through a consensus mechanism, eliminating reliance on a centralized trusted third party. Key characteristics include:

- Decentralization—data is redundantly stored across all network nodes, with no single point of failure;
- Immutability—altering any historical block requires recalculating all subsequent blocks and controlling a majority of computational power, incurring prohibitively high theoretical and economic costs;
- Transparent verifiability—transaction records are publicly accessible and traceable;
- State determinism—all compliant nodes reach consensus on the current state of the ledger.

This architecture provides a trusted execution environment and persistent state storage foundation for smart contracts, with its security directly dependent on the robustness of the underlying blockchain's consensus mechanism and cryptographic guarantees.

B. Smart Contracts: Definition and Execution Model

Smart contracts are autonomous programs deployed on blockchain platforms[16], where business logic and execution conditions are encoded in software and executed automatically upon fulfillment of predefined triggering criteria, without requiring human intervention. Ethereum enables smart contracts to handle complex state transitions and cross-contract interactions through the introduction of the Turing-complete Ethereum Virtual Machine (EVM)[17]. The typical development workflow involves writing source code using high-level languages such as Solidity or Vyper, compiling it into platform-specific bytecode (EVM bytecode)[18], and submitting a deployment transaction to write it onto the blockchain, thereby generating a unique contract address.

During execution, external accounts or other contracts initiate call transactions, and the EVM interprets bytecode instructions sequentially within an isolated sandbox environment, consuming gas to perform computations and updating on-chain states accordingly[19]. Core characteristics include determinism (identical inputs invariably produce identical outputs), non-termination under valid calls (continuous responsiveness), and immutability of deployed code. The latter constitutes a critical security constraint: once a logical flaw exists within a contract, remediation via hotfixes is impossible, leaving vulnerabilities permanently exposed to potential exploitation.

C. Standardized Smart Contract Vulnerability Taxonomy

Smart contract vulnerabilities refer to defects in code logic that can be exploited to cause asset loss, state inconsistency, or privilege escalation. This paper adopts the Smart Contract Weakness Classification Registry as a unified taxonomy benchmark[20]. Maintained under the leadership of Consensys Diligence and aligned with the CWE framework, this standard possesses both community consensus and formal descriptive capability[21]. Definitions of core vulnerability types are as follows:

- Reentrancy (SWC-107): An external call triggers a callback before internal state updates are completed, leading to repeated execution of critical logic;
- Arithmetic overflow/underflow (SWC-104): Unsigned integer operations exceed the type range, causing numeric wraparound;
- Access control flaws (SWC-105): Critical functions lack permission checks such as require(msg.sender == owner);
- Timestamp dependency (SWC-116): Business logic directly relies on block.timestamp, making it susceptible to miner-induced manipulation;
- Unchecked call return values (SWC-100): Failures from low-level calls such as call or send are not detected, resulting in inconsistent state.

D. Vulnerability Detection Metrics.

The vulnerability detection task is formally defined as follows[22]: given a sequence of EVM bytecode instructions $I=\{op1,op2,\ldots,opn\}$ (where opidenotes an opcode), the model outputs a binary label vector $Y=\{y1,y2,\ldots,ym\}$, where m

Kun Cheng and Sun Zhang, "A Review of AI-Driven Approaches for Vulnerability Detection in Smart Contract Bytecode," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 8, Issue 10, pp. 47-52, 2026.

represents the set of predefined vulnerability types and yj=1 indicates that I contains a vulnerability of type j. In industrial scenarios, it is also necessary to locate the positions of vulnerable instructions, corresponding to a sequence labelling task.

Evaluation strictly employs the following metrics:

- Recall: measures the ability to control missed detections, and holds the highest priority in security contexts;
- Precision and F1 score: balance detection accuracy against the high cost of false positives;
- False positive rate (FPR): FPR=FP+TNFP, which directly impacts the workload of manual verification.

E. Benchmark Resources.

Mainstream datasets include:

- SmartBugs Wild: comprising 47,800 mainnet contracts covering ten SWC vulnerability categories, with annotations derived from historical audit reports and manual verification;
- ContractBench: a cross-chain dataset spanning Ethereum, BSC, and Polygon, containing 12,350 contracts and providing standardized bytecode along with vulnerability location annotations;
- Juliet SC: a synthetic test suite incorporating controllable vulnerability variants and boundary cases, designed for robustness validation of detection methods.

The contract scale, vulnerability distribution density, annotation protocols, and applicability boundaries of each dataset are summarized in Table I (presented in the main text). This section objectively presents the task definition, quantitative metrics, and resource parameters to establish a reproducible evaluation context for method comparisons in Section Three.

III. METHODLOGY

This section systematically reviews current mainstream deep learning-based detection methods targeting smart contract bytecode. According to the structured representation of bytecode and the architectural characteristics of the models, existing work is categorized into distinct approaches, each of which clearly defines its input construction logic, core model design, quantitative performance on standard benchmarks, and applicable boundaries.

A. Graph Neural Network Frameworks

Zhuang et al. proposed an event-driven temporal graph neural network (ETGN) that integrates node execution order, edge types, and timestamps into an event sequence, enhancing the representation of event concurrency through a self-attention mechanism[23]. The model applies BERT pre-training encoding graph nodes, incorporating bidirectional contextual semantics, and achieves 94.67% accuracy in detecting reentrancy vulnerabilities across 120,932 real-world contracts, representing a 9.09% improvement over traditional GNNs. ETGN's event generation module (EGM) maps execution time to periodic nonlinear representations via Time-to-Vector (T2V) encoding, effectively capturing temporal dependencies between cross-function invocations. Experiments show that retaining the EGM increases the average recall rate for three classes of vulnerabilities by 10.62%, confirming the critical value of temporal information in representing complex vulnerability semantics.

Zhu et al. further introduced a semantic-aware heterogeneous graph network (SAHGN), distinguishing between control-dependency edges, data-dependency edges, and function-call edges, and designing a multi-relational aggregation function to fuse heterogeneous neighbor information. In cross-contract reentrancy detection tasks[24],SAHGN identifies semantically dispersed fragments across multiple contracts through message passing, achieving an F1 score of 92.4%, an 8.3 percentage point improvement over homogeneous graph models. Current GNN-based methods still face challenges such as dependence on graph construction quality and high computational overhead: errors in parsing dynamic jumps in bytecode can lead to graph structure discontinuities, and graphs generated from large DeFi protocols often contain over 2,000 nodes, with single inference times reaching 2.3 seconds, limiting deployment for real-time detection. Emerging efforts are exploring lightweight GNN architectures and incremental graph construction strategies to balance accuracy and efficiency.

B. Convolutional Architectures for Opcode Patterns

Hwang et al. designed the SVCB model, which deploys CNN and BiLSTM dual pathways in parallel: the CNN branch employs three layers of convolution with kernel sizes of 3, 5, and 7 to extract local opcode patterns, while the BiLSTM branch captures global dependencies across execution sequences[25]. Feature vectors from both branches are concatenated and subsequently weighted by an attention mechanism to emphasize critical vulnerability features.

On the ETH dataset, SVCB achieves an average accuracy of 91.70% for four categories of vulnerabilities, which is a 4.23 percentage point improvement over a standalone CNN. Specifically, for reentrancy vulnerability detection, the attention layer focuses on the critical gap between CALL instructions and state updates, raising the recall rate to 93.45%. To address data imbalance, the model incorporates SMOTE oversampling, which synthesizes samples for each label column independently. This technique increases the F1 score for minority-class vulnerabilities (e.g., Tx.origin) from 76.21% to 88.74%.

Despite its performance, the parallel architecture has limitations, including increased inference latency (approximately 0.38 seconds per contract) and sensitivity to input sequence length. Memory consumption in the feature fusion layer grows nonlinearly when the number of opcodes exceeds 10,000. Recent work aims to mitigate these issues through channel pruning and knowledge distillation, reducing the parameter count by 67% while maintaining over 90% accuracy, offering a promising direction for resource-constrained scenarios.

Experiments demonstrate that CNN-based architectures are highly effective at detecting pattern-fixed vulnerabilities, such

Kun Cheng and Sun Zhang, "A Review of AI-Driven Approaches for Vulnerability Detection in Smart Contract Bytecode," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 8, Issue 10, pp. 47-52, 2026.

as timestamp dependency and unchecked call return values. In contrast, detecting logical vulnerabilities that require long-range semantic reasoning necessitates the integration of sequential models to achieve enhanced performance.

C. Sequence Modeling Approaches

Temporal models transform normalized bytecode into instruction sequences and capture implicit dependencies within program execution paths through recurrent neural networks or attention mechanisms. Gautrin et al. proposed a masked attention-RNN hybrid architecture that first constructs a control flow graph (CFG) from bytecode[26], retains only pure opcode instructions via node cleaning, and then applies three categories of vulnerability-oriented masks—memory/call-related, time/gas-related, and arithmetic/logic-related—to generate differentiated subgraphs. These subgraphs are vectorized via Graph2Vec and fed into an RNN temporal model for joint inference.

This approach achieves 96% accuracy on the BCCC-VulSCs-2023 dataset, with a recall rate of 94% for reentrancy vulnerabilities, significantly outperforming single-representation models. Its innovation lies in the masking mechanism dynamically focusing on critical semantic regions and avoiding interference from irrelevant instructions; for example, when detecting reentrancy vulnerabilities, it automatically increases the weights of instructions such as CALL, MSTORE, and SSTORE, enhancing the model's sensitivity to malicious callback paths by 32.7%.

Experiments demonstrate that the internal memory mechanism of RNNs effectively preserves vulnerability signals across basic blocks, particularly in long-sequence scenarios (contract bytecode exceeding 5,000 bytes), yielding an 8.4 percentage point improvement in F1 score compared with CNN architectures.

D. Large Language Models and Agent-Based Frameworks

Large language models (LLMs), leveraging their robust code comprehension capabilities, have opened new avenues for smart contract vulnerability detection[27]; however, direct application faces two primary limitations: existing LLMs achieve an F1 score of only approximately 41.1% in specialized audit workflows, and their performance varies substantially across different vulnerability types (e.g., reentrancy, integer overflow). Agent4Vul addresses these constraints through a multimodal agent architecture, introducing two types of LLM-driven agents: the Commentator agent employs a flexible prompt strategy library (FPSL) to generate code comments, encompassing five strategies—simple description, detailed description, role-playing, chain-of-thought (CoT) reasoning, and vulnerability-specific CoT (e.g., focusing on balance variables and call.value usage for reentrancy vulnerabilities); the Vectorizer agent transforms source code and comments into high-dimensional vectors, capturing semantic features incrementally via CoT reasoning.

This framework innovatively integrates a semantic branch (source code plus comments) with a graph branch (bytecode CFG), achieving detection F1 scores ranging from 92.53% to 98.58% for four common vulnerability types on a dataset of over 40,000 real-world contracts, significantly outperforming advanced LLMs such as GPT-4o (13.36%–34.29%) and o1 (3.73%–34.29%). Its principal breakthrough lies in translating LLM cognitive capabilities into actionable vulnerability representations, rather than relying solely on singular prompt engineering.

E. Multi-modal and Specialized Architectures

Multimodal fusion architectures overcome the semantic limitations of single-model approaches by synergizing features from distinct representation spaces. COBRA innovatively integrates bytecode semantic context with function interface information to construct an interaction-aware vulnerability detection framework**Error! Reference source not found.**. Its core comprises four components: a semantic extraction module that converts bytecode into static single assignment (SSA) format opcode sequences; a function signature recovery module (SRIF) that infers undisclosed function parameter types and quantities from control flow graphs using a seq2seq structure; a path sequence construction module that concatenates trigger semantics with interface features; and a training module for joint learning of vulnerability patterns. SRIF achieves an F1 score of 94.76% on 99,745 function signatures, addressing the industry challenge in which 15.62% of contracts lack publicly available ABIs.

In vulnerability detection tasks, when original ABIs are available, COBRA attains an F1 score of 93.45%; in ABI-missing scenarios, it maintains a recall rate of 89.46% by inferring function characteristics. Its key innovation lies in the latent space fusion of function interfaces and semantic context—through the MS-CAM module, it captures both local features of function attributes (e.g., the association between payable modifiers and callvalue instructions) and global features (permission propagation across function call chains), precisely locating semantic discontinuities between missing msg.sender checks and call.value invocations in reentrancy vulnerability detection.

IV. Challenges and Future Directions

Deep learning methodologies have demonstrated notable enhancements in the precision of bytecode vulnerability detection within laboratory settings; however, the transition of these methodologies into industrial-grade security tools continues to confront a range of systemic challenges. Current research predominantly concentrates on optimizing metrics for individual benchmark tests, yet it inadequately addresses the dynamic complexity inherent in real-world blockchain ecosystems: the continual evolution of vulnerability patterns, the escalating diversity across cross-chain platforms, and the stringent demands of security audits for interpretability and minimal false alarm rates.

A. Data-Centric Bottlenecks

The quality and distribution of labeled datasets represent fundamental constraints for existing approaches. Prominent datasets such as SmartBugs Wild and ContractBench manifest significant class imbalance: reentrancy vulnerability samples

Kun Cheng and Sun Zhang, "A Review of AI-Driven Approaches for Vulnerability Detection in Smart Contract Bytecode," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, Volume 8, Issue 10, pp. 47-52, 2026.

constitute over 42% of the total, whereas complex types like logic bypass and cross-contract composition vulnerabilities account for less than 5%. This skewed distribution directly induces a bias in the model's decision boundary towards prevalent vulnerabilities; for instance, the CNN-BiLSTM model attains an F1 score of 94.3% in integer overflow detection, yet its recall rate for logic vulnerabilities plummets to 68.2%.

Synthetic datasets (e.g., Juliet SC) offer controllable vulnerability variants, but their instruction patterns diverge semantically from real-world attacks. Zhang et al. confirmed that models trained on synthetic data with a recall rate exceeding 90% experience a performance decline of over 25 percentage points when deployed on mainnet contracts. The absence of standardized annotation protocols across cross-chain platforms further intensifies data fragmentation issues: bytecode structural disparities between BSC and Polygon contracts diminish the average cross-chain F1 score of a singular model by 11.7%. More critically, reliance on manual annotation introduces substantial variability contingent on expert factors. Cross-validation based on ContractBench reveals an annotation noise rate as high as 18.3%. Additionally, the evolution of vulnerability definitions, exemplified by the inclusion of the "oracle manipulation" category in the SWC registry in 2024, swiftly renders historical annotations obsolete.

B. Model Generalization Barriers

The strong reliance of models on historical vulnerability patterns makes them fragile in dynamic adversarial environments. Although Agent4Vul achieves an F1 score exceeding 92% in detecting known vulnerabilities, its recall for the newly disclosed “flash loan combined reentrancy” variant in 2025 is only 53.8%, revealing a deficiency in zero-shot generalization capability. Bytecode decompilation noise emerges as a latent impediment to generalization: when the control flow graph (CFG) construction error rate exceeds 15%—a condition frequently encountered in DeFi contracts with dynamic jumps—the performance degradation of GNN-based methods reaches 22.4%, significantly higher than that observed in sequential models.

The black-box nature of decision-making further erodes industrial trust: security engineers face difficulties in verifying the basis for VulBERTa’s “high-risk” assessments, leading 83% of audit teams to still conduct manual reviews of all positive results (Immunefi 2025 Industry Report). Cross-platform migration also encounters architectural gaps: models optimized for EVM bytecode experience an average F1 score decrease of 19.6% when directly applied to WASM bytecode, reflecting the deeper challenge of insufficient semantic abstraction levels across bytecode formats.

C. Evaluation and Reproducibility Gaps

The absence of standardization and insufficient transparency in the current evaluation framework significantly hinder objective measurement of method performance and the efficiency of technological iteration. Benchmark datasets frequently undergo adjustments in vulnerability distributions and variations in annotation granularity during version evolution, undermining the comparability of research outcomes across different time periods. Reporting of metrics exhibits a clear selective bias: most studies emphasize single advantageous indicators such as recall, while discussions concerning critical industrial deployment parameters—including false positive rates and computational overhead—remain relatively sparse, limiting the comprehensive assessment of tool practicality.

Disclosure of experimental details also suffers from widespread deficiencies; the omission of hyperparameters, preprocessing rules, and randomness control elements poses substantial obstacles to independent replication. Crucially, existing evaluation frameworks lack robustness verification mechanisms tailored to real-world threat scenarios—such as adversarial test sets constructed via semantics-preserving instruction perturbations or cross-compiler variants—resulting in a cognitive gap between laboratory accuracy and actual efficacy in complex on-chain environments. Establishing a unified, transparent, and industrially relevant evaluation ecosystem has thus become an urgent prerequisite for advancing technologies from academic innovation to practical security deployment.

V. Conclusion

Currently, AI-powered solutions continue to confront a multitude of challenges. This paper offers a systematic review of the developmental trajectory of deep learning techniques within the realm of smart contract bytecode vulnerability detection. Empirical analyses demonstrate that, in benchmark tests like SmartBugs Wild and ContractBench, these methodologies consistently attain F1 scores surpassing 90% when identifying prevalent vulnerabilities such as reentrancy and integer overflow, markedly outperforming traditional static analysis tools in terms of detection efficacy.

Nevertheless, the refinement of technical metrics does not invariably elevate the maturity level of industrial applications. Challenges including class imbalance and annotation noise at the data level, model reliance on historical vulnerability patterns, and the fragmented nature of the evaluation ecosystem collectively constitute substantial bottlenecks in ongoing research endeavors. The significance of this study extends beyond merely tracing the methodological evolution; it also elucidates the interplay between accuracy, generalization capability, and credibility. Insights such as the trade-off between high recall and elevated false positive rates furnish empirical evidence for tool developers.

Acknowledgement

This work was supported by the Humanity and Social Science Research Project of Anhui Educational Committee (Grant No. 2024AH052138), the University-Industry Collaborative Education Program of the Ministry of Education (Grant No. 231100751213206), the Open Research Fund of Key Laboratory of River Basin Digital Twinning of Ministry of Water Resources (Grant No. KLRBDT202507), and the Natural Science Youth Research Project of the Department of

Education of Anhui Province (Grant No. 2025AHGXZK40443).

REFERENCES

[1] M. Nofer, P. Gomber, O. Hinz, et al., “Blockchain,” Bus. Inf. Syst. Eng., vol. 59, no. 3, pp. 183–187, 2017.
[2] W. Zou, D. Lo, P. S. Kochhar, Y. Zhou, J. Xu, and Y. Liu, “Smart contract development: Challenges and opportunities,” IEEE Trans. Softw. Eng., vol. 47, no. 10, pp. 2084–2106, 2019.
[3] G. O. Young, “Synthetic structure of industrial plastics,” in Plastics, 2nd ed., vol. 3, J. Peters, Ed. New York: McGraw-Hill, pp. 15–64, 1964.
[4] W.-K. Chen, Linear Networks and Systems. Belmont, CA: Wadsworth, pp. 123–135, 1993.
[5] G. Zao, “When deep learning meets smart contracts,” in 35th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE), Canberra, Australia, pp. 1400–1402, 2020.
[6] J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility,” IEEE Transactions Electron Devices, vol. ED-11, no. 1, pp. 34–39, 1959.
[7] E. P. Wigner, “Theory of traveling-wave optical laser,” Physical Review, vol. 134, pp. A635–A646, 1965.
[8] P. Zhu, J. Hu, X. Li, and Q. Zhu, “Using Blockchain Technology to Enhance the Traceability of Original Achievements,” IEEE Trans. Eng. Manage., vol. 70, no. 5, pp. 1693–1707, 2023.
[9] H. Rezaei, A. A. Monrat, K. Andersson, and F. Palmieri, “TaintSentinel: Path-Level Randomness Vulnerability Detection for Ethereum Smart Contracts,” in IEEE Int. Conf. Blockchain, Zhengzhou, China, pp. 15–22, 2025.
[10] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, “Empirical review of automated analysis tools on 47,587 Ethereum smart contracts,” in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20), New York, NY, USA, pp. 530–541, 2020.
[11] N. Harrand, C. Soto-Valero, M. Monperrus, and B. Baudry, “The Strengths and Behavioral Quirks of Java Bytecode Decompilers,” in 19th Int. Working Conf. on Source Code Analysis and Manipulation (SCAM), Cleveland, OH, USA, pp. 92–102, 2019.
[12] D. E. Siegel, “Understanding The DAO Attack,” CoinDesk, 2018.
[13] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Ethereum Yellow Paper, pp. 1–32, 2014.
[14] Z. Li, S. Lu, R. Zhang, “Vulhunter: Hunting vulnerable smart contracts at EVM bytecode-level via multiple instance learning,” IEEE Transactions on Software Engineering, vol. 49, no. 11, pp. 4886–4916, 2023.
[15] A. J. Deva Priya Isravel, K. M. Sagayam, B. Bhushan, Y. Sei, and J. Eunice, "Blockchain for healthcare systems: Architecture, security challenges, trends and future directions," Journal of Network and Computer Applications, vol. 215, p. 103633, 2023.
[16] A. P. Balcerzak, E. Nica, E. Rogalska, M. Poliak, T. Klieštik, and O.-M. Sabie, "Blockchain Technology and Smart Contracts in Decentralized Governance Systems," Administrative Sciences, vol. 12, no. 3, p. 96, 2022.
[17] V. Capocasale and G. Perboli, "Standardizing Smart Contracts," IEEE Access, vol. 10, pp. 91203-91212, 2022..
[18] X. Su, H. Liang, H. Wu, B. Niu, F. Xu, and S. Zhong, "DiSCo: Towards Decompiling EVM Bytecode to Source Code using Large Language Models," in Proceedings of the ACM Software Engineering, vol. 2, no. FSE, Article FSE103, pp. 1-24, July 2025.
[19] S. Siddiqui, S. Hameed, S. A. Shah, A. K. Khan, and A. Aneiba, “Smart contract-based security architecture for collaborative services in municipal smart cities,” Journal of Systems Architecture, vol. 135, p. 102802, 2023.
[20] M. Ortu, G. Ibba, G. Destefanis, C. Conversano, and R. Tonelli, "Taxonomic insights into ethereum smart contracts by linking application categories to security vulnerabilities," Scientific Reports, vol. 14, no. 23433, 2024.
[21] F. R. Vidal, N. Ivaki, and N. Laranjeiro, "OpenSCV: an open hierarchical taxonomy for smart contract vulnerabilities," Empirical Software Engineering, vol. 29, no. 101, 2024.
[22] S. Jeon, G. Lee, H. Kim, and S. S. Woo, "Design and evaluation of highly accurate smart contract code vulnerability detection framework," Data Mining and Knowledge Discovery, vol. 38, no. 3, pp. 888-912, 2024.
[23] Wang, K., Zhang, W,andTu, M., "CGCN-DF: A Cascade GCN Vulnerability Detection Framework for Secure CIoT Smart Contracts," IEEE Transactions on Consumer Electronics, early access, 2026.
[24] S. T. Gandhi, “AI-Driven Smart Contract Security: A Deep Learning Approach to Vulnerability Detection,” International Journal of Advanced Research in Computer Science & Technology (IJARCST), vol. 8, no. 1, pp. 11540–11547, 2025.
[25] S. J. Hwang, S. H. Choi, J. Shin, “CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection,” IEEE Access, vol. 10, pp. 32595–32607, 2022.
[26] M. Gautrin, A. Habibi Lashkari, and S. HajiHosseinKhani, “A deep learning-based vulnerability detection in blockchain smart contracts using masked attention and control flow graph analysis,” Int. J. Inf. Secur., vol. 24, Art. No. 240, 2025.
[27] J. W. Jie, W. Qiu, and H. Yang, “Agent4Vul: multimodal LLM agents for smart contract vulnerability detection,” Science China Information Sciences, vol. 68, no. 6, Art. No. 160101, 2025.
[28] X. Zenggang, L. Qiangqiang, and L. Youfeng, “NDLSC: A new deep learning-based approach to smart contract vulnerability detection,” Journal of Signal Processing Systems, vol. 97, no. 1, pp. 49–68, 2025.