Mesopotamian journal of Big Data
Vol. **(2025)**, 2025, **pp.** 178–194
DOI: https://doi.org/10.58496/MJBD/2025/012 , ISSN: 2958-6453
https://mesopotamian.press/journals/index.php/BigData

Review Article

AI-Driven Smart Contract Vulnerability Detection: A Systematic Review of Methods, Challenges, and Future Prospects

Saad AL Azzam[1,*], Raenu AL Kolandaisamy [1], Ghassan AL Dharhani[1]

[1] *Institute of Computer Science and Digital Innovation, UCSI University, Kuala Lumpur, Malaysia*

ARTICLEINFO

Article History

Received 28 Jun 2025
Revised 25 Jul 2025
Accepted 11 Aug 2025
Published 30 Aug 2025

Keywords

Artificial Intelligence
Sustainable Growth
Security analysis
Deep Learning
Machine Learning
Smart contracts
Blockchain
Vulnerability detection
Ethereum
Sustainable Digital Infrastructure
Sustainable AI-Blockchain Integration

ABSTRACT

Smart contracts (SCs) have become an essential component in the world of decentralized applications, automating transactions across blockchain networks without the need for intermediaries, and with this rise in adoption, the technology has also brought forth growing concern due to security vulnerabilities, which have led to serious financial damage, and the problem is far from being solved. Traditional auditing methods often struggle to capture the more intricate vulnerabilities hidden within smart contract logic, particularly owing to the irreversible nature of blockchain transactions. Given these challenges, researchers have been actively exploring more advanced detection techniques. Despite progress, many existing studies tend to focus narrowly on specific methods, whether static analysis, dynamic testing, or machine learning models, without offering a comprehensive comparison across all available approaches. This fragmented landscape leaves a noticeable gap for practitioners looking for a well-rounded understanding of smart contract security solutions. To address this, our study set out to systematically review the existing body of work, analysing 21 reviewed studies published between 2020 and 2024. The primary aim was to combine the diverse techniques that have been proposed for detecting vulnerabilities in smart contracts, ranging from static and dynamic analyses to more recent AI-driven models, graph-based techniques, and hybrid systems, critically evaluating their strengths, weaknesses, and practical effectiveness. The methodology followed a structured approach. We searched major research databases, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus—using carefully crafted search queries to ensure that we captured the most relevant and up-to-date papers. Our findings revealed that AI-based methods, especially those leveraging deep neural networks and graph neural networks, have achieved impressive detection accuracy in controlled environments. For example, models such as ContractWard and SCVDIE-ENSEMBLE reported Micro-F1 scores of 98.48% and 95.46%, respectively, but these models also have a trade-off—they demand high computational resources, which limits their real-world deployment in resource-constrained settings. On the other hand, lighter tools such as Slither and NeuCheck offer faster detection and lower resource usage but might fall short in regard to identifying more complex or new vulnerabilities. We also noticed a growing trend towards real-time monitoring tools, such as SODA and GPTScan, which aim to strike a balance by reducing false positives while providing proactive security measures. However, several challenges remain unresolved where many AI-driven models still rely heavily on labelled datasets, which may not generalize well to novel attack patterns. Scalability is another concern, especially for models that are computationally intensive.

1. INTRODUCTION

With the emergence of blockchain technology and its rapid development, it has led to radical transformations in many sectors, such as the industrial and financial sectors, by relying on this technology on decentralized transaction systems that prioritize security in executing transactions, which enhances transparency and increases levels of security [1] [2]. Smart contracts (SCs) are programs written in specialized languages that execute automatically, eliminating intermediaries, as the mechanism for executing these contracts is through predefined rules that are written via dedicated programming instructions so that they meet the desired goal [3]. The programming languages used to write SCs are high-level programming languages, examples of which are Solidity, Vyper, and Rust [4]. SC is used to automate processes in many areas, such as financial exchanges, where it is currently used to exchange digital currencies such as Bitcoin and Ethereum, supply chains and identity

*Corresponding author. Email: Snazzam.199@gmail.com verification. Therefore, SC is vulnerable to exploitation by attackers, who can exploit flaws in the written code that may be caused by human error [5].

Detecting vulnerabilities in SC systems is a significant challenge because of their complex nature, where traditional auditing methods struggle to address these complexities because there is no central point of control. Furthermore, transactions are considered irreversible after publication, making it difficult to detect security vulnerabilities early or address them after publication [6].

Deploying SCs in blockchain (BC) applications can lead to catastrophic problems and billions of dollars in financial losses if they contain security vulnerabilities [7], where these vulnerabilities can arise for many reasons, including (1) immature programming languages [8], (2) a lack of development experience [9], and (3) insufficient development and verification tools [10].

These issues highlight the need for measures that are adaptive to the rapid developments taking place [11]. In response to these problems, researchers have developed various techniques, such as static analysis, dynamic analysis, machine learning (ML), and deep learning (DL), to uncover these vulnerabilities [12] [13] [14], each of which has advantages and disadvantages.

Many tools have emerged to solve these problems in SCs, such as Mythril [15], Slither [16], and Oyente [17]. However, there are still limitations, including scalability limitations and difficulty identifying new or complex vulnerabilities.

This study conducts a comprehensive review of current research on SC vulnerability detection, focusing on current trends, challenges, and future directions that can contribute to enhancing the security of BC systems.

Over the past few years, numerous surveys and review articles have explored different methods for detecting vulnerabilities in smart contracts. For example, [12] examined formal verification and static analysis techniques in their study. [10] also reviewed static and dynamic analysis tools; however, they did not delve into AI-driven approaches. Moving forward, [13] contributed a review that highlighted machine learning methods, but their work did not address graph-based models or the need for real-time monitoring solutions.

While these efforts have certainly advanced the field, there is still a noticeable gap. No comprehensive study has successfully combined static, dynamic, machine learning/deep learning, graph-based, and real-time detection strategies under one unified framework. Moreover, some of the latest advancements, such as transformer-based architectures and zero-shot learning techniques, are scarcely mentioned in the current literature.

The primary objectives are (1) to identify and classify state-of-the-art detection techniques; (2) to critically analyse their strengths, weaknesses, and applicability; and (3) to highlight emerging trends and unresolved challenges. From these objectives, the following research questions are addressed:

- What are the predominant methods currently used for smart contract vulnerability detection, and how do they differ in scope and performance?
- What gaps or limitations exist in current detection techniques, and how might they be addressed in future research?
- How can AI-based and hybrid approaches be optimized for both accuracy and scalability in real-world deployments?

This paper aims to bridge these gaps. We present a systematic review that captures a broad spectrum of detection methodologies published between 2020 and 2024. In addition to listing tools, we provide a detailed comparative analysis of their performance. Additionally, the paper offers critical insights into emerging trends and outlines directions for future research in this rapidly evolving domain.

The remainder of this paper is organized as follows: Section 2 presents the background on blockchain and smart contract fundamentals; Section 3 outlines the methodology of the review; Section 4 provides a detailed literature analysis; Section 5 offers a comparative evaluation and discussion; and Section 6 concludes the paper with key findings and future research directions.

2. BACKGROUND

2.1 BC Basic Architecture

The BC uses a sequential data structure, where requests are raised for different blocks according to their chronological age. The first block is known as the 'genesis block'. The information of a transaction is a data structure that represents the transfer of value through signature processes. A block represents a collection of transaction data and consists of a block header and a block body. The block header includes [18]:

- Protocol Version: This version is utilized for software update tracking.
- Previous Block Hash: It records the previous block hash value in the current block. This unique hash value, generated through an irreversible hash algorithm based on the block's information, serves to uniquely identify the block. Storing the hash value of the previous block in the current block ensures a link between the two blocks.
- Merkle Root: Records the hash value of the Merkle tree root of this block.
- Timestamp: The timestamp of the block creation is recorded. This timestamp ensures that the blockchain's data can be stored in chronological order, allowing the data's origin to be traced on the basis of the block's timestamp.
- Difficulty Target: Indicates the difficulty coefficient that must be solved for the current block.
- Nonce: A dynamically calculated value derived through computational effort during the mining process.

The block body contains the content of the transaction and the relevant information. Each transaction is highlighted with a digital signature, which ensures block data security. Typically, the block body includes the following components:

- Transaction Byte Size: Indicates the storage space occupied by transactions.
- Transaction Count: Denotes the total number of transactions in the block.
- Transaction Data: Contains the actual transactional records bundled within the block.

All transaction information within the block's body is organized via the Merkle tree structure represented in Fig. 1 [19]. This information is located in the leaf nodes. The leaf nodes are hashed together to generate successive hash values, which continue to combine until the Merkle tree root node is obtained [20].

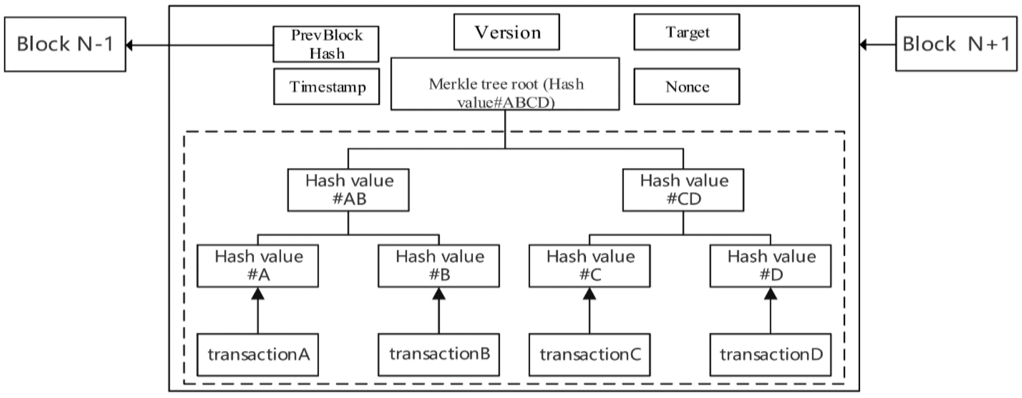

Fig 1. Merkle tree structure

2.2 BC mechanism

The BC process can be divided into three stages: (1) block formation, (2) consensus verification, and (3) ledger maintenance.

- Block Formation: During this phase, network nodes gather information about the transactions broadcast across them. These nodes compete for the privilege of validating these transactions on the basis of the node's computing power. Nodes that obtain validation rights receive incentives according to a specific reward system. These incentives are used to encourage nodes to continually contribute their computing resources to the BC network.
- Consensus Validation: At this stage, the node that has obtained transaction verification rights sends this block (after verification) to the BC network, and all the participating nodes in the network receive this block and verify its content

on the basis of the consensus protocol used in the BC network. These nodes evaluate the block content and record it on the BC ledger.

- Ledger maintenance: This stage is called ledger maintenance or ledger updating. The network nodes that receive the block in step 2 are stored in the BC ledger for long-term preservation. These nodes then perform periodic verifications via the timestamp and block hash value to ensure that the blocks in the ledger have not been tampered with. This process enables applications at higher layers to query the ledger information.

2.3 BC structure

Fig. 2 illustrates the structure of the BC layers [21]:

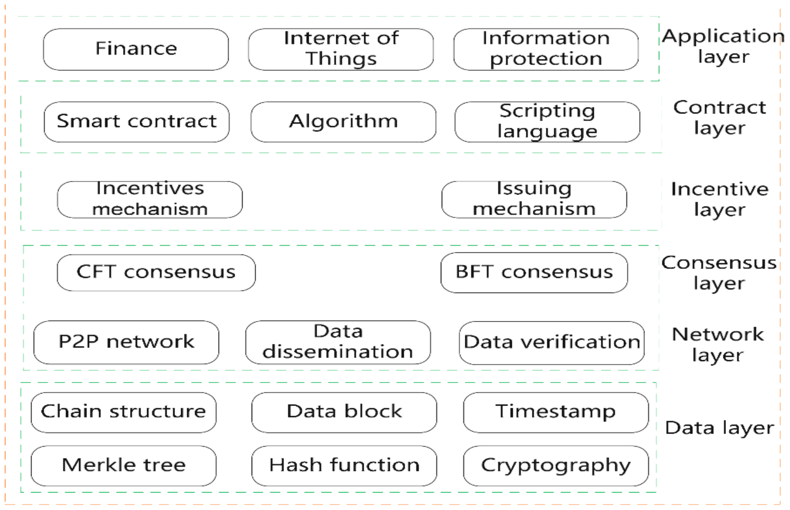

Fig 2. The structure of the BC layers [21]

The data layer includes the chain structure, data blocks, timestamps, Merkle trees, hash functions, and cryptographic methods. This layer forms the basis for various functions of BC, such as management, organization, and storage. The network layer uses a peer-to-peer (P2P) mechanism to connect all nodes within the chain. A consensus algorithm determines which nodes have the right to execute and verify transactions. The incentive layer integrates rewards and penalties into the BC system. The smart contract layer contains the set of contracts that are being executed. The application layer combines the base structure, code script, and SC, allowing BC to be applied to various real-world scenarios [7].

2.4 Smart contract

Smart contracts are self-executing programs used to automatically implement the terms of agreements between parties, where these contracts enable the execution of the agreement's terms once predefined conditions are met, reducing the potential for manipulation or disputes between parties. The SC life cycle consists of four critical stages: creation, deployment, execution, and completion [22].

- Stage 1-Creation: At this stage, the SC code is written.
- Stage 2-Deployment: Deploy the SC to the Ethereum Virtual Machine (EVM) where it will be executed.
- Stage 3-Execution: At this stage, the smart contract is actively engaged in processing the incoming transactions and any accompanying data where the virtual machine methodically executes the code line by line until the task is complete or the allocated gas limit has been reached. This sequence unfolds as the network proceeds to mine the next block.
- Stage 4-Completion: In this stage, the contract’s state is updated and recorded on the BC ledger with its associated transactions.

3. METHODOLOGY

To perform a comprehensive and unbiased evaluation of SC vulnerability detection techniques, a systematic literature review was implemented. The procedure consisted of identifying relevant research databases, creating focused search queries, and applying predetermined inclusion and exclusion criteria to filter the final set of studies. This study sought to collect, organize, and analyse contemporary tools, models, and frameworks that support the detection of SC vulnerabilities on multiple BC platforms.

We utilized the following databases for data collection: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect (Elsevier), and Scopus.

To identify relevant papers, a combination of keywords was used. The primary keywords used were as follows:

- SC vulnerabilities.
- SC security.
- BC vulnerability detection.
- Ethereum SC attacks.
- AI in SC security
- SC vulnerability detection deep neural network (DNN).
- SC vulnerability detection (ML).

The search was conducted between [02/2025] and [04/2025] using the carefully selected keywords mentioned above. Additional filters restricted the results to peer-reviewed publications in English published between January 2020 and December 2024, and we also performed backwards and forward citation tracking to identify relevant studies not captured in the initial search. Duplicate entries were removed via EndNote, followed by manual verification.

The initial search retrieved 412 studies. After duplicates were removed, 356 studies remained. The titles and abstracts of the searches excluded 292 studies that did not meet the inclusion criteria, leaving 64 studies for the full review. Among these studies, 43 were excluded because of limited methodological details, a lack of empirical results, or a lack of relevance to the scope of the search, resulting in 21 studies being excluded from the final analysis. Fig 3 shows the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) flow diagram.

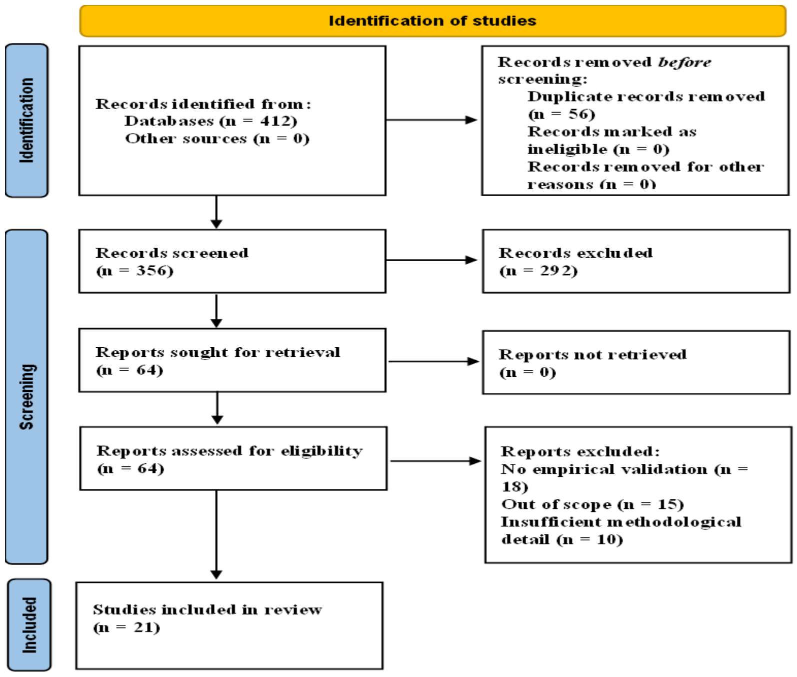
Identification of studies
Identification
Records identified from:
Databases (n = 412)
Other sources (n = 0)
Records removed *before* screening:
Duplicate records removed (n = 56)
Records marked as ineligible (n = 0)
Records removed for other reasons (n = 0)
Screening
Records screened (n = 356)
Records excluded (n = 292)
Reports sought for retrieval (n = 64)
Reports not retrieved (n = 0)
Reports assessed for eligibility (n = 64)
Reports excluded:
No empirical validation (n = 18)
Out of scope (n = 15)
Insufficient methodological detail (n = 10)
Included
Studies included in review (n = 21)

Fig 3. The PRISMA flow diagram

Each study included in the review was evaluated via the following quality indicators:

- The direct focus is on smart contract vulnerability detection within blockchain systems.
- Clarity and completeness of the proposed detection approach.
- Availability of quantitative performance metrics (accuracy, precision, recall, F1 score).
- Sufficient detail provided to enable replication of the study.
- Introduction of new techniques or significant improvements over existing methods.

Only studies meeting at least four of the five criteria were included.

Publications were included in the analysis if they met the specified criteria:

- Published between 2020 and 2024 (to focus on recent advancements in BC security).
- Focus on SC vulnerabilities.
- Security issues in the BC-based SC are discussed.
- Presents new detection methods, tools, or frameworks for SC vulnerability detection.

4. LITERATURE REVIEW

In this section, we analyse the techniques and tools used to detect vulnerabilities in SCs. The literature is divided into three categories: (1) static analysis tools (STAs), (2) dynamic analysis tools, and (4) ML and DNN tools. Fig. 4 shows the yearwise development of analysis tools for BC-based SC.

4.1 SATs

The authors of [23] presented a tool called Ethainter, which is an SAT used to sensor complex vulnerabilities in SCs. Traditional security analysis tools have focused on individual vulnerabilities, but Ethainter has expanded its detection capabilities to include multiple vulnerabilities that can appear in an SC. Ethainters track how untrusted inputs spread in smart contracts, and models bypass mechanisms for protection conditions. An ethainter is an SAT that relies on rules stored in a data log. These rules are used to analyse the logic of SC execution. The ethainter achieved an overall accuracy of 82.5%.

Reference [24] presented a formal symbolic process virtual machine (FSPVM-E), a hybrid formal verification system designed to ensure the reliability and security of Ethereum-based SC, which combines symbolic execution, theorem proving, and static analysis to detect vulnerabilities at the source code level.

FSPVM-E consists of four core components:

- GERM (a general, extensible, and reusable memory framework)
- Lolisa (a formal Solidity specification language)
- Father (a formal interpreter at Coq)
- Helper tools and libraries that detect basic vulnerabilities and reduce workload.

The authors of [25] presented a tool called NeuCheck, a SAT based on the syntax tree to detect vulnerabilities in SC that relies on symbolic execution and dependency graphs and uses syntactic analysis to detect vulnerabilities faster. NeuCheck works in three stages: (1) converting the source code to a syntax tree, (2) using DOM4j, an XML parser, to query the syntax tree for security patterns, and (3) generating reports that identify the type and location of the vulnerability. NeuCheck outperforms symbolic execution-based tools in speed and scalability and supports multiple platforms (Windows, Linux, and Mac).

Reference [26] presented a secure Ethereum smart contract (SESCon), an SAT that improves SC security by detecting vulnerabilities on the basis of predefined patterns. SESCon applies bug analysis to trace data flows and identify security risks and then applies XML path (Xpath) queries to detect vulnerabilities at the syntax tree level. SESCon consists of three core modules: (1) a vulnerable patterns module to extract standard vulnerability patterns from Ethereum security documentation; (2) an XPath module that converts the Solidity code to an abstract syntax tree (AST) using XPath queries to detect common vulnerabilities; and (3) a contamination analysis module that tracks changes to state variables, function dependencies, and data flows. SESCon achieves 90% detection accuracy

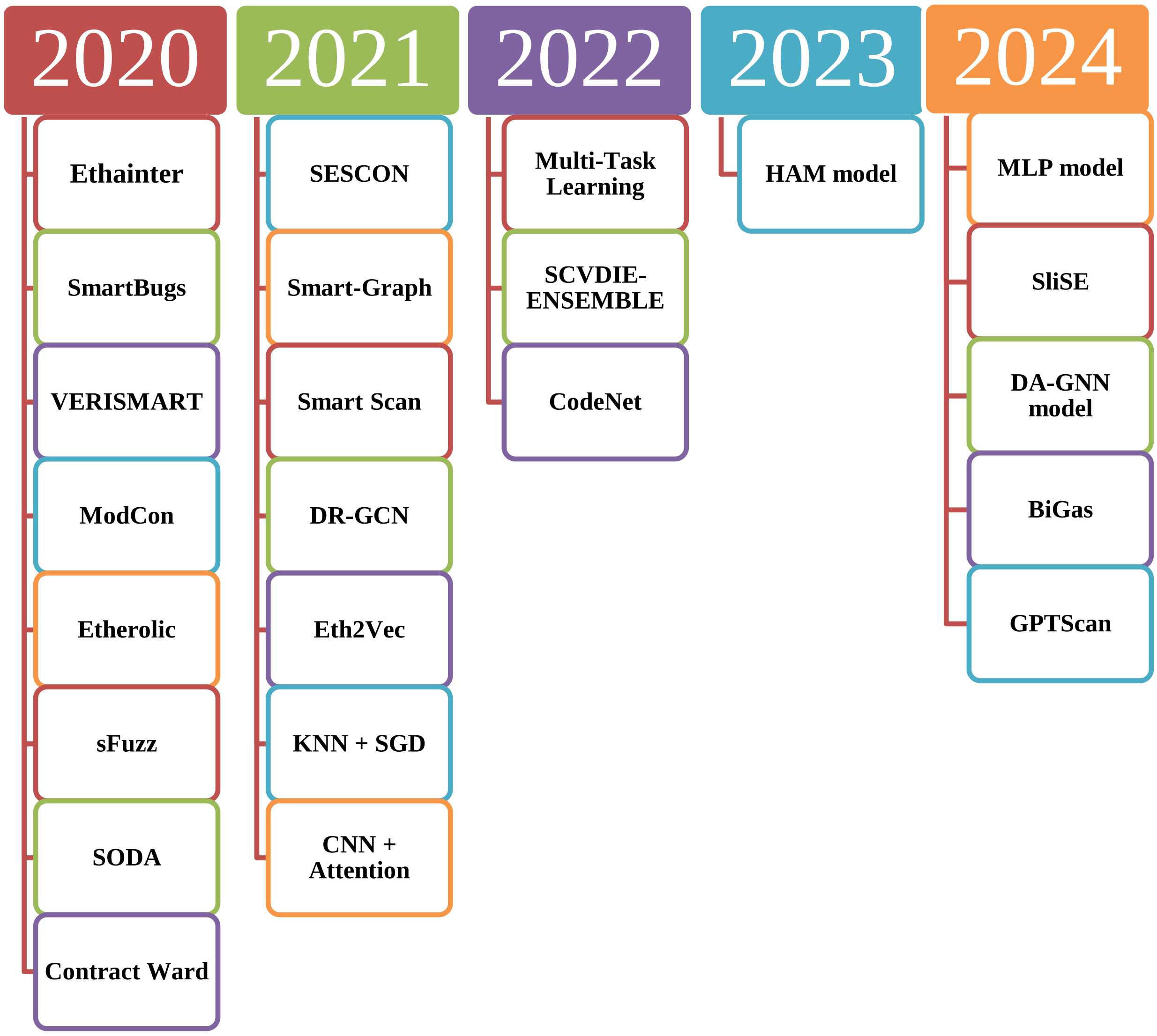

Fig 4. The annual progression of analytical tools designed for BC-based SC

Reference [27] presented the SmartBugs framework, a framework used to analyse the Solidity SC. SmartBugs simplifies automated security analysis by integrating 10 security analysis tools, namely, Slither, Mythril, and SmartCheck, and provides selected datasets for security assessment. SmartBugs consists of five main components: (1) a command-line interface (CLI) used to run security tools on SC, (2) tool configurations used to specify execution parameters for each tool, (3) Docker-based security tools that use Docker images for standard execution environments, (4) curated Solidity datasets that provide labelled SCs with known vulnerabilities, and (5) a SmartBugs Runner that coordinates analysis tools and collects security reports. SmartBugs first converts Solidity SC to an AST representation and then runs multiple security tools to detect vulnerabilities and generate security reports for comparison. Mythril achieves the highest detection rate at 27%. Ninety-seven percent of real-world contracts were flagged as vulnerable, indicating a high false positive rate. SmartBugs provide the largest experimental setup for solidity security evaluation and facilitate security tool integration and comparison.

Reference [28] introduces Smart-Graph, a web-based tool that generates graphical representations of SC. The Smart-Graph provides Unified Modelling Language (UML) diagrams for SCs that visualize the interactions of functions and contracts within decentralized applications (DApps). In a smart graph, users enter the address of an SC into the web interface, and the tool retrieves the contract source code from Etherscan, parses the contract code, extracts structural elements, and generates a graphical representation in the form of UML diagrams that show large or complex functions. A smart graph has been applied to real-world contracts and has demonstrated better readability and clarity than text-based contract analysis does, thus enhancing SC debugging by clarifying dependencies.

Research [29] presents SmartScan, a tool used to detect denial of service (DoS) vulnerabilities caused by unexpected rollbacks in Ethereum SC that occur due to a DOS attack when a contract fails due to deliberate transaction rollback, rendering it inoperable. SmartScan operates in three stages. The first stage is the static analysis stage, where pattern matching and the transformation language are used to identify vulnerable sending, transfer, and communication functions. The second stage is the dynamic analysis stage, where the SC under test is deployed on a private Ethereum test network. The third stage is the reporting stage, where the type of vulnerability is determined.

Research [30] has presented the verifier smart (VERISMART), which is an SC security tool. This tool focuses on computational security. VERISMART uses a verification algorithm consisting of 4 stages: (1) Transaction-specific constant discovery, where conditions that apply to all smart contract transactions are automatically discovered; (2) static loop analysis ensures safe execution of repetitive functions; (3) comprehensive path analysis, which evaluates all possible execution paths; and (4) an improved satisffiability modulus (SMT) analyser, which uses advanced techniques to efficiently handle computational limitations. Table 1 shows a comparative summary of static analysis tools (SATs).

TABLE I. A COMPARATIVE SUMMARY OF STATIC ANALYSIS TOOLS (SATS)

Ref	**Year**	**Detection Method**	**Vulnerabilities Covered**	**Key Strengths**	**Limitations**
[23]	2020	Static Analysis	-	Highly precise taint analysis, Real-world attack validation	Cannot detect new vulnerability types not modelled in its rules
[24]	2020	symbolic execution, theorem proving, and static analysis	-	Ensures correctness of Solidity contracts before deployment	it requires familiarity with Coq.
[25]	2021	syntax tree-based static analysis	Access Control Vulnerability, Reentrancy Attacks, Hash Collision, Integer Overflow/Underflow, Dependence on Predictable Variables	Outperforms symbolic execution-based tools, with Cross-platform support (Windows, Linux, Mac).	Can't detect new Vulnerabilities
[26]	2021	Static Analysis based on Taint analysis and XPath queries	SWC-100, SWC-107, SWC-113, SWC-120, SWC-122	Combines XPath queries & taint analysis to reduce false positives, analyses real-world Ethereum contracts with high accuracy.	It is limited to predefined vulnerability patterns and can't detect new Vulnerabilities.
[27]	2020	Multi-Tool Framework	DASP10 Vulnerabilities	Integrates 10 security tools, a dataset for benchmarking	It is limited to predefined vulnerability patterns and can't detect new Vulnerabilities
[28]	2021	graphical representations of Ethereum SC	-	Improves contract understanding, visualizes dependencies.	No vulnerability detection
[29]	2021	Static + Dynamic Analysis	DoS via Unexpected Revert	Combining pattern matching and testing on private networks	Limited to DoS vulnerabilities
[30]	2020	Formal Verification	Arithmetic Safety, Overflows, Underflows	Lowest false positive rate, exhaustive verification	Limited to arithmetic safety, it cannot detect new vulnerabilities or vulnerabilities that were previously unknown to the tool

4.2 Dynamic analysis

Reference [31] introduces model-based contracts (ModCon), which allows developers to define expected contract behavior via state machines and supports state transitions, pre/post conditions, and system constants. The tool is capable of generating test cases via different strategies to ensure state testing, state transitions, and complex interactions. The tool has a graphical interface that allows users to upload contracts, define models, and execute tests.

Research [32] has presented Ethereum concolic (Etherolic), a security analyser based on obfuscation. Etherolic consists of six components: (1) a contamination engine that flags untrusted inputs and tracks their propagation, (2) an attack indicator that detects security violations during execution, (3) a security detector that determines whether existing security mechanisms mitigate vulnerabilities, (4) a log analyser that collects runtime data for debugging and reporting, (5) a test generator that generates inputs to confirm vulnerabilities, and (6) a report generator that produces vulnerability reports. Ethereum runs on EVM bytecode. The results have shown that it reduces the number of false positives.

Reference [33] introduces a solidity fuzzer (sFuzz), a feedback-driven adaptive testing tool for SCs. The tool relies on adaptive camouflage. It uses American fuzzy lag (AFL)-inspired mutation strategies to generate test inputs. The tool also monitors the execution process to detect vulnerabilities.

The authors of [34] presented a smart contract online detection attack (SODA) framework. It is a framework for detecting attacks on EVM-compatible SCs in real time. SODA collects information to extract transaction execution data and then relies on a detection layer through which custom detection applications can be created for real-time monitoring to detect suspicious transactions. This model features a compatibility mechanism that enables applications to run on multiple BC platforms without modification. This framework is limited to EVM-compatible BC platforms. Furthermore, detection applications must be manually developed for new types of vulnerabilities, and these applications are susceptible to human error and poor implementation, making them vulnerable to breaches and attacks. Table 2 shows a comparative summary of the dynamic analysis tools.

TABLE II. A COMPARATIVE SUMMARY OF DYNAMIC ANALYSIS TOOLS

Ref	Year	Detection Method	Vulnerabilities Covered	Key Strengths	Limitations
[31]	2020	Model-Based Testing	Business Logic Flaws	Best for enterprise contracts, generates high-quality test cases	Requires manual modelling
[32]	2020	Concolic Execution + Fuzzing	integer overflow/underflow, reentrancy, and short address attacks	Works without source code, generates exploits	Manual verification is needed for some vulnerabilities
[33]	2020	Adaptive Fuzzing	Gasless Send, Exception Disorder, Reentrancy Attacks, Timestamp & Block Number Dependency, Dangerous Delegatecall, Integer Overflow/Underflow, Freezing Ether	Fastest SC fuzzer, multiobjective optimization	It cannot detect new and emerging vulnerabilities
[34]	2020	Real-Time Monitoring	Reentrancy Attacks, Unexpected Function Invocation, Short Address Attack, Tx.origin Authentication Exploits, Unchecked External Calls, Missing Transfer Events, Block Number & Timestamp Dependence	Low overhead, works across EVM BC s	Detection applications must be manually developed for new types of vulnerabilities

4.3 DL/ML-based methods

The study presented in [35] benefited from graph-based representations and graph neural networks (GNNs). The experiments used 3000 SCs. The methodology involves converting SC code into graphs that represent the syntactic and semantic structures discovered within the SC code. Master nodes represent function calls, and secondary nodes represent variables. This model achieved an accuracy of 84.48%.

The authors of [36] presented a tool called ContractWard, which is based on ML, to detect six vulnerabilities in SC. The study used 49,502 SC. Five classifiers were tested: eXtreme Gradient Boosting (XGBoost), Adaptive Boosting (AdaBoost), Random Forest, Support Vector Machine (SVM), and k-nearest neighbors (KNN). XGBoost achieved the best performance (Micro-F1= 98.48% and Macro-F1= 96.41%).

Study [37] presented Eth2Vec. Eth2Vec then uses the distributed memory model of paragraph vectors (DM-PV) model to learn code representations and then converts the code sequences into vector representations via natural language processing (NLP) techniques. The average accuracy achieved by Eth2Vec is 77.0%.

In accordance with [38], a multitask learning approach was implemented to improve the detection of smart contract vulnerabilities. This approach consists of a lower sharing layer, where an attention mechanism combined with an artificial neural network (ANN) encodes contract text into vectors, and a task-specific layer, where convolution neural networks

(CNNs) perform vulnerability classification. The system reported detection accuracies of 77.5% for computational vulnerabilities, 70.31% for reentry vulnerabilities, and 78.85% for cases involving unknown addresses.

The study in [39] introduces an ML-based approach for detecting vulnerabilities in Ethereum SC. The methodology begins with a feature extraction phase to extract key features, which are then used to train the CNN and random forest algorithms. The study focuses on bytecode analysis, and the authors acknowledge that source code analysis can provide deeper insights.

The study in [40] introduced the smart contract vulnerability detection method based on information graph and ensemble learning (SCVDIE-ENSEMBLE), an SC vulnerability detection model that integrates multiple neural networks to increase accuracy and robustness. Using a dataset of 21,667 Ethereum SCs (11,756 vulnerable and 9,911 healthy), the methodology involves converting contracts into opcode sequences, generating information graphs (IGs) to capture relationships, and embedding features via Word to Vector (Word2Vec), Global Vectors for Word Representation (GloVe), and FastText. Seven neural networks, a CNN, a recurrent neural network (RNN), a region-based convolutional neural network (RCNN), a DNN, a gated recurrent unit (GRU), a bidirectional gated recurrent unit (Bi-GRU), and transformers, are trained, and their outputs are combined through ensemble learning (EL). The model achieves 95.46% accuracy and an F1 score of 97.57%.

The study in [41] introduced CodeNet, a custom CNN-based model for detecting vulnerabilities in SCs. The methodology involves preprocessing SC code by compiling it into bytecode, converting it into a fixed-size format, and mapping it into an image for CNN analysis. CodeNet achieves 97.66% accuracy. However, the fixed-size image representation may lead to information loss in large SCs, and the nonscalable CNN design increases computation and memory demands, potentially causing performance bottlenecks on resource-limited devices.

The study proposed by [42] proposes an ML-based approach for detecting vulnerabilities in Ethereum SC, focusing on security risks that have led to major financial losses. The methodology uses ASTs to identify structural similarities between vulnerable and nonvulnerable contracts. KNN and stochastic gradient descent (SGD) are employed for vulnerability detection. The KNN model achieved 90% accuracy.

The study in [43] presented a DL approach that combines CNNs with an attention mechanism. The model uses CNNs with an attention mechanism to extract features from SC bytecode. The attention mechanism improves the model by capturing long-term dependencies between code sequences. After feature extraction, CNNs process the code sequences, and a SoftMax layer performs final classification to identify vulnerabilities. The model was tested on a dataset of 8,632 SCs, achieving 85% accuracy.

Reference [44] presented an approach to detect vulnerabilities in SCs via a multilayer perceptron (MLP-ANN)-based ML model. The authors propose a tool that benefits from opcode and control flow graphs (CFGs) for feature extraction. The authors balanced the dataset through a fault injection method. The opcode and CFG features are embedded via 3D vectors and the term frequency-inverse document frequency (TFIDF). Although fault injection is a good addition for balancing the dataset and avoiding biased models, relying on artificial human vulnerability injection may lead to gaps in capturing real-world vulnerability patterns. This limitation can affect the robustness of the model in real-time scenarios. The validity of the research can be enhanced by relying on sampling techniques to balance the dataset, which is based on creating synthetic samples on the basis of the analysis of existing samples. The research compares the proposed model with existing static and dynamic analysis tools, and no comparison is made with other ML or DL models, so including models such as transformer-based architectures could provide better performance.

The authors of [45] presented a tool called slicing symbolic execution (SliSE) designed to detect reentry vulnerabilities in SCs. The entry vulnerabilities are as follows: Program slicing is used to examine the intercontract program dependency graph (I-PDG) of an SC, generating warnings about potential vulnerabilities. SliSE reached an F1 score of 78.65%. The paper does not discuss how the proposed tool handles large or complex contracts. The symbolic execution can be very expensive in terms of resources and may have scaling issues, especially as the contract size increases.

Research [46] presented the Domain Adaptive Graph Neural Networks (DA-GNN) model for detecting vulnerabilities in SCs. The DA-GNN applies a dual attention mechanism within the graph attention network to extract features from the contract code. The model works well on special types of vulnerabilities and does not address other vulnerabilities or new vulnerabilities that may arise with the continuous development of attack mechanisms and techniques. GNNs are computationally expensive and require high amounts of resources, especially when working on large and complex SCs.

Research [47] presented a model called BiGAS, which is a model for detecting reentry vulnerabilities in SC. This name refers to the main components on which it is built: bidirectional GRU + attention + SVM (BiGAS). The authors replace the

SoftMax classifier with SVM. The model is designed to detect only reentry, which limits its applicability across a wider range of vulnerabilities.

Reference [48] presented a hybrid attention mechanism (HAM) model that detected five types of vulnerabilities. The approach consists of three main phases:

- Code Fragment Extraction: At this stage, the SC source code is analysed to extract code fragments that potentially contain security vulnerabilities.
- Training phase: The model uses both single-head and multihead attention encoders to capture different aspects of the code's semantic and contextual information.
- Finally, the model employs a fully connected network to classify the code fragments as vulnerable or not.

Reference [49] introduces a model called GPTScan, which is an SC vulnerability detection model that includes the capabilities of generative pretrained transformers (GPTs) with static code analysis. The tool divides the detection process into three main steps:

- The GPT is used to match the initial scenario and potential vulnerability characteristics.
- Static analysis was used to confirm the GPT results.
- The irrelevant functions are filtered out before the GPT step to reduce the computational overhead.

GPTScan was evaluated on three datasets (Top200, Web3Bugs, and DefiHacks), achieving a false positive rate of 4.39% and a recall value of 70%. Although GPTScan was tested on three datasets, these datasets have a limited range of vulnerabilities. Thus, its ability to detect new or emerging types of vulnerabilities remains unclear. Implementing mechanisms to learn from evolving vulnerabilities would ensure that GPTScan remains relevant as SC attack strategies evolve.

Table 3 shows a comparative summary of machine learning and deep learning methods.

TABLE III. A COMPARATIVE SUMMARY OF MACHINE LEARNING AND DEEP LEARNING METHODS

Ref	**Year**	**Detection Method**	**Vulnerabilities Covered**	**Key Strengths**	**Limitations**
[35]	2021	Graph Neural Networks (GNNs)	Reentrancy, Timestamp Dependence, Infinite Loops	Uses graph-based contract representations, with high accuracy	High computational cost, requires training data
[36]	2020	ML (XGBoost, RF, SVM)	Timestamp Dependence, Reentrancy, Overflow/Underflow, Call Stack Depth, Transaction Order Dependence	High detection accuracy, tested on 49,502 contracts	Limited to 6 vulnerability types, cannot detect new vulnerabilities
[37]	2021	ML + NLP (PV-DM)	SC Bytecode-Level Vulnerabilities	Learns from bytecode without predefined features	Struggles with unknown vulnerabilities
[38]	2022	CNN + Attention-Based Neural Network	Computational Vulnerabilities, Reentrancy, Unknown Address Issues	Captures contextual meaning, improves accuracy	Hard parameter sharing weakens generalization
[39]	2020	Neural Network (NNBOOF, CNNBOSM, RFBOOF)	Opcode-Level Vulnerabilities	Uses opcode analysis, supports multiple detection models	Lacks dataset balancing, only bytecode-level analysis
[40]	2022	Ensemble Learning (CNN, RNN, RCNN, DNN, GRU, Bi-GRU, Transformers)	Opcode Sequences, Contract Relationships	High accuracy using multiple neural networks	High computational cost
[41]	2022	Custom CNN	SC Vulnerabilities	Optimized for feature extraction, outperforms VGGNet	High computation/memory demand, image representation may cause information loss
[42]	2021	KNN + SGD	Structural Similarities in SC	Uses ASTs to find vulnerabilities	No dataset balancing may introduce bias
[43]	2021	CNN + Attention	SC Bytecode-Level Vulnerabilities	Captures long-term dependencies, NLP-based	Limited to 5 vulnerability types

[44]	2024	Multi-Layer Perceptron (MLP)	Opcode Features, CFG-Based Detection	Standardized preprocessing, balanced dataset with synthetic errors	Synthetic errors may introduce bias
[45]	2024	Program Slicing + I-PDG Analysis	Reentrancy Vulnerabilities	Uses program slicing for intercontract analysis	Limited to reentrancy, scaling issues for large contracts
[46]	2024	Graph Neural Networks (GNNs) + Dual Attention	SC Vulnerabilities	Improved feature extraction for contract analysis	High computational demand
[47]	2024	Bi-GRU + SVM	Reentrancy Vulnerabilities	Improved classification accuracy	Limited to reentrancy
[48]	2023	Hybrid Attention + Neural Network	5 SC Vulnerabilities	Captures both single and multihead attention	No dataset balancing, limited to 5 vulnerabilities
[49]	2024	GPT + Static Code Analysis	SC Vulnerability Detection	Uses GPT for initial detection, static analysis for confirmation	Limited dataset, unclear detection for new vulnerabilities

5. KEY FINDINGS, GAPS, AND FUTURE DIRECTIONS

On the basis of the literature reviewed, there is an increasing reliance on ML and graph-based models to detect SC vulnerabilities. Studies such as ContractWard and SCVDIE-ENSEMBLE demonstrate the effectiveness of ensemble learning and DNNs in identifying SC vulnerabilities. Graph-based methods such as the graph-based GNN, DA-GNN, and SliSE show strong performance in modelling SCs as interconnected nodes, which significantly improves SC vulnerability detection by capturing patterns and relationships between code fragments. Additionally, runtime monitoring and intrusion detection tools such as Sereum, SODA, and GPTscan are emerging as viable solutions that help reduce false positives and provide proactive security.

Studies such as Eth2Vec and GPTScan, which use natural language processing (NLP) techniques to analyse SC code, have shown strong capabilities in capturing patterns and relationships within code, emphasizing the importance of leveraging context to detect vulnerabilities and leveraging attention-based transformers and networks to improve detection performance. Adaptive fuzzing techniques such as CrossFuzz, sFuzz, and EVMFuzzer can address security issues at the EVM level rather than at the code level. Hybrid approaches that combine static and dynamic analysis (e.g., ModCon, SolAnalyser) have also proven to be more effective in capturing different types of vulnerabilities.

Notably, many ML-based tools, such as Eth2Vec, ContractWard, and SCVDIE-ENSEMBLE, struggle to detect unknown or emerging vulnerabilities because they rely on prelabelled datasets that may not generalize well. Furthermore, the high computational costs associated with graph-based and DNN models such as DA-GNN, BiGAS, and CodeNet pose scalability issues for real-world deployment. Another critical gap is the limited focus on interactions between contracts, as most studies evaluate individual contracts rather than analysing how vulnerabilities propagate across multiple contracts. Additionally, real-time performance monitoring tools such as SODA and GPTscan are still limited by false positives and reliance on predefined rule sets.

Several critical gaps emerged from this synthesis. First, most AI-driven models rely heavily on labelled datasets, which restricts their ability to detect previously unseen vulnerabilities and limits generalizability across different blockchain platforms. Second, while graph-based models improve structural analysis, they are computationally intensive and present scalability challenges, particularly for real-time applications. Third, the majority of tools focus on vulnerabilities within individual contracts, with limited research into how vulnerabilities propagate across interacting contracts. Fourth, real-time monitoring frameworks are often designed for EVM-compatible platforms, leaving non-EVM blockchains underexplored. Finally, there is a lack of standardized evaluation benchmarks, making it difficult to compare tools fairly or replicate results across studies.

To address these gaps, future research should focus on discovering new or emerging vulnerabilities by developing adaptive AI models that can identify unknown vulnerabilities without relying solely on predefined datasets. Graph-based detection models and DNN architectures can also be improved to reduce computational overhead and improve real-time applicability; thus, future work could focus on combining symbolic execution, static analysis, and DNNs to improve detection accuracy and reduce false positives.

When comparing the available tools, ContractWard and SCVDIE-ENSEMBLE stood out for their impressive accuracy, where they achieved Micro-F1 scores of 98.48% and 95.46%, respectively. However, this level of precision comes at a price because models such as SCVDIE-ENSEMBLE, which rely on cluster-based architectures, demand significant computational resources, making them less practical for real-time detection scenarios.

On the other hand, tools such as Slither and NeuCheck—both lightweight static analysis frameworks—are incredibly efficient where they require minimal computational power and deliver rapid analysis. However, the trade-off is that these tools often struggle to maintain high accuracy, especially when confronted with intricate or unconventional vulnerabilities.

There are runtime monitoring solutions such as SODA, which offer middle ground by balancing resource usage and detection performance. However, manual intervention is typically needed to develop and fine-tune detection applications.

In summary, balancing accuracy, resource efficiency, and detection speed across various smart contract vulnerability detection methods is challenging.

Table 4 shows the rankings of the SC vulnerability detection tools.

TABLE IV. RANKING OF SC VULNERABILITY DETECTION TOOLS

TOOL NAME	DETECTION ACCURACY (SCORE)	COMPUTATIONAL RESOURCES (SCORE)	DETECTION SPEED (SCORE)	TOTAL SCORE	RANK
CONTRACTWARD	High	Medium	High	**8**	**1**
NEUCHECK	Medium	High	High	**8**	**1**
SLITHER	Medium	High	High	**8**	**1**
SODA	Medium	Medium	High	**7**	**2**
SCVDIE-ENSEMBLE	High	Low	Medium	**6**	**3**
ETHAINTER	Medium	Medium	Medium	**6**	**3**
DA-GNN	High	Low	Low	**5**	**4**
SMARTSCAN	Low	High	Medium	**6**	**3**
ETHEROLIC	Medium	Medium	Low	**5**	**4**
GPTSCAN	Medium	Medium	Medium	**6**	**3**
FSPVM-E	High	Low	Low	**5**	**4**
SMARTBUGS	Medium	Low	Medium	**5**	**4**
VERISMART	High	Low	Low	**5**	**4**
CODENET	High	Low	Low	**5**	**4**
SFUZZ	Medium	Medium	High	**7**	**2**

6. CRITICAL ANALYSIS OF METHODS STRENGTHS AND WEAKNESSES

Graph-based methods effectively model smart contracts as node-link structures to capture detailed intercode relationships, but they are computationally expensive and struggle to scale for large or complex contracts.

Machine learning (ML) and deep neural networks (DNNs) often achieve high accuracy in detecting predefined vulnerabilities within training datasets, but they struggle with generalizing to unseen or emerging vulnerabilities owing to their dependency on labelled datasets.

By analysing code without execution, static analysis enables the efficient detection of recurring vulnerability patterns. However, it is less effective in capturing flaws involving multiple contracts or vulnerabilities that become apparent only through dynamic behavior.

Dynamic analysis tools, while capable of simulating runtime behaviors to uncover complex attack scenarios, often require manual effort in creating test cases or state models.

Hybrid approaches that combine static and dynamic analysis show promise in improving detection accuracy and coverage, but they introduce additional complexity and computational overhead.

7. CONCLUSIONS

This systematic review provides an in-depth analysis of the latest developments in smart contract vulnerability detection, drawing on a diverse body of research encompassing static and dynamic analysis tools, machine learning and deep learning techniques, and emerging hybrid and graph-based approaches. By examining 21 studies published between 2020 and 2024, the review identified the most effective detection strategies and the key limitations of each methodological category.

From a comparative perspective, the results reveal a consistent trend: hybrid and AI-based models tend to outperform traditional methods in terms of detection accuracy, particularly when identifying known vulnerabilities. In particular, graph-based methods demonstrate a strong ability to capture the structural and contextual relationships embedded in smart contract code—patterns often overlooked by other techniques. However, this ability comes at a cost. Their computational requirements can be significant, making them less suitable for immediate deployment. In contrast, lightweight static analysis tools offer speed and lower resource consumption, but they may struggle to detect new or highly complex vulnerabilities.

This study contributes in three ways. First, it brings together a fragmented field into a coherent, codified framework encompassing multiple methodological areas. Second, it provides a critical performance comparison, accurately identifying the intersections of accuracy, scalability, adaptability, and trade-offs. Third, it identifies critical research gaps and proposes practical paths to address them, paving the way for progress in this rapidly evolving field.

The implications of these findings extend to both theory and practice. From a theoretical perspective, this review enriches the understanding of how methodological decisions affect vulnerability detection outcomes, providing a solid foundation for future comparative studies and model innovation. Practically, it provides a guide for developers, auditors, and security analysts to select tools that are appropriate for specific operational contexts—whether high-precision detection in resource-rich environments or rapid, lightweight scanning in situations where time is essential. More importantly, recognizing ongoing challenges, such as the urgent need for adaptive models capable of identifying zero-day vulnerabilities, sets clear priorities for the next generation of tools.

Finally, this work not only maps the current state of the field but also lays the foundation for methodologies that balance accuracy, efficiency, and adaptability. By linking theoretical insights with practical application, this work supports the creation of more secure, scalable, and resilient blockchain systems in the face of evolving threats.

Conflicts of interest

The authors have no relevant financial or nonfinancial interests to disclose.

Funding

The authors declare that no funds, grants, or other support was received during the preparation of this manuscript.

Acknowledgement

The first draft of the manuscript was written by the authors. No external writing assistance was provided. All final edits and intellectual contributions were made solely by the authors, and no external funding or payments were involved in the writing process.

References

[1] M. Javaid, A. Haleem, R. P. Singh, S. Khan, and R. Suman, "Blockchain technology applications for Industry 4.0: A literature-based review," Blockchain: Research and Applications, vol. 2, no. 4, p. 100027, 2021.

[2] M. Javaid, A. Haleem, R. P. Singh, R. Suman, and S. Khan, "A review of Blockchain Technology applications for financial services," BenchCouncil transactions on benchmarks, standards and evaluations, vol. 2, no. 3, p. 100073, 2022.

[3] F. K. H. Mihna et al., "Bridging Law and Machine Learning: A Cybersecure Model for Classifying Digital Real Estate Contracts in the Metaverse," Mesopotamian Journal of Big Data, vol. 2025, pp. 35-49, 2025.

[4] A. Bakhshi, Securing Smart Contracts: Strategies for Identifying and Mitigating Vulnerabilities in Blockchain Applications. University of Central Arkansas, 2024.

[5] D. Kirli et al., "Smart contracts in energy systems: A systematic review of fundamental approaches and implementations," Renewable and Sustainable Energy Reviews, vol. 158, p. 112013, 2022.

[6] G. Zheng, L. Gao, L. Huang, and J. Guan, Ethereum smart contract development in solidity. Springer, 2021.

[7] K. Hu, J. Zhu, Y. Ding, X. Bai, and J. Huang, "Smart contract engineering," Electronics, vol. 9, no. 12, p. 2042, 2020.

[8] W. Zou et al., "Smart contract development: Challenges and opportunities," IEEE transactions on software engineering, vol. 47, no. 10, pp. 2084-2106, 2019.

[9] A. Singh, R. M. Parizi, Q. Zhang, K.-K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," Computers & Security, vol. 88, p. 101654, 2020.

[10] V. Rishiwal, U. Agarwal, M. Yadav, A. Alotaibi, P. Yadav, and S. Tanwar, "Blockchain-Secure Gaming Environments: A Comprehensive Survey," IEEE Access, 2024.

[11] M. Almakhour, L. Sliman, A. E. Samhat, and A. Mellouk, "Verification of smart contracts: A survey," Pervasive and Mobile Computing, vol. 67, p. 101227, 2020.

[12] B. Nirmala, R. Abueid, and M. A. Ahmed, "Big Data distributed support vector machine," Mesopotamian Journal of Big Data, vol. 2022, pp. 12-22, 2022.

[13] Y. Xu, T. Slaats, B. Düdder, T. Troels Hildebrandt, and T. Van Cutsem, "Safe design and evolution of smart contracts using dynamic condition response graphs to model generic role-based behaviors," Journal of Software: Evolution and Process, vol. 37, no. 1, p. e2730, 2025.

[14] N. Sharma and S. Sharma, "A survey of Mythril, a smart contract security analysis tool for EVM bytecode," Indian Journal of Natural Sciences, vol. 13, no. 75, pp. 51003-51010, 2022.

[15] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019: IEEE, pp. 8-15.

[16] L. JJ and K. Singh, "Enhancing Oyente: four new vulnerability detections for improved smart contract security analysis," International Journal of Information Technology, vol. 16, no. 6, pp. 3389-3399, 2024.

[17] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in 2017 IEEE international congress on big data (BigData congress), 2017: Ieee, pp. 557-564.

[18] R. C. Merkle, "A digital signature based on a conventional encryption function," in Conference on the theory and application of cryptographic techniques, 1987: Springer, pp. 369-378.

[19] L. Peng, W. Feng, Z. Yan, Y. Li, X. Zhou, and S. Shimizu, "Privacy preservation in permissionless blockchain: A survey," Digital Communications and Networks, vol. 7, no. 3, pp. 295-307, 2021.

[20] H. Xiong, M. Chen, C. Wu, Y. Zhao, and W. Yi, "Research on progress of blockchain consensus algorithm: A review on recent progress of blockchain consensus algorithms," Future Internet, vol. 14, no. 2, p. 47, 2022.

[21] Y. Liu et al., "An overview of blockchain smart contract execution mechanism," Journal of Industrial Information Integration, vol. 41, p. 100674, 2024.

[22] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: a smart contract security analyzer for composite vulnerabilities," in Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, 2020, pp. 454-469.

[23] Z. Yang, H. Lei, and W. Qian, "A hybrid formal verification system in coq for ensuring the reliability and security of ethereum-based service smart contracts," IEEE Access, vol. 8, pp. 21411-21436, 2020.

[24] N. Lu, B. Wang, Y. Zhang, W. Shi, and C. Esposito, "NeuCheck: A more practical Ethereum smart contract security analysis tool," Software: Practice and Experience, vol. 51, no. 10, pp. 2065-2084, 2021.

[25] A. Ali, Z. U. Abideen, and K. Ullah, "SESCon: Secure Ethereum smart contracts by vulnerable patterns' detection," Security and Communication Networks, vol. 2021, no. 1, p. 2897565, 2021.

[26] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "Smartbugs: A framework to analyze solidity smart contracts," in Proceedings of the 35th IEEE/ACM international conference on automated software engineering, 2020, pp. 1349-1352.

[27] G. A. Pierro, "Smart-graph: Graphical representations for smart contract on the ethereum blockchain," in 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021: IEEE, pp. 708-714.

[28] N. F. Samreen and M. H. Alalfi, "Smartscan: an approach to detect denial of service vulnerability in ethereum smart contracts," in 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2021: IEEE, pp. 17-26.

[29] S. So, M. Lee, J. Park, H. Lee, and H. Oh, "Verismart: A highly precise safety verifier for ethereum smart contracts," in 2020 IEEE Symposium on Security and Privacy (SP), 2020: IEEE, pp. 1678-1694.

[30] Y. Liu, Y. Li, S.-W. Lin, and Q. Yan, "ModCon: A model-based testing platform for smart contracts," in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1601-1605.

[31] M. Ashouri, "Etherolic: a practical security analyzer for smart contracts," in Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020, pp. 353-356.

[32] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "sfuzz: An efficient adaptive fuzzer for solidity smart contracts," in Proceedings of the ACM/IEEE 42nd international conference on software engineering, 2020, pp. 778-788.

[33] T. Chen et al., "SODA: A Generic Online Detection Framework for Smart Contracts," in NDSS, 2020.

[34] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," in Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence, 2021, pp. 3283-3290.

[35] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Blockchain-enabled authentication handover with efficient privacy protection in SDN-based 5G networks," IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 1133-1144, 2021.

[36] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts," in Proceedings of the 3rd ACM international symposium on blockchain and secure critical infrastructure, 2021, pp. 47-59.

[37] J. Huang, K. Zhou, A. Xiong, and D. Li, "Smart contract vulnerability detection model based on multi-task learning," Sensors, vol. 22, no. 5, p. 1829, 2022.

[38] C. Xing, Z. Chen, L. Chen, X. Guo, Z. Zheng, and J. Li, "A new scheme of vulnerability analysis in smart contract with machine learning," Wireless Networks, vol. 30, no. 7, pp. 6325-6334, 2024.

[39] L. Zhang et al., "A novel smart contract vulnerability detection method based on information graph and ensemble learning," Sensors, vol. 22, no. 9, p. 3581, 2022.

[40] S.-J. Hwang, S.-H. Choi, J. Shin, and Y.-H. Choi, "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," IEEE Access, vol. 10, pp. 32595-32607, 2022.

[41] Y. Xu, G. Hu, L. You, and C. Cao, "A novel machine learning-based analysis model for smart contract vulnerability," Security and Communication Networks, vol. 2021, no. 1, p. 5798033, 2021.

[42] Y. Sun and L. Gu, "Attention-based machine learning model for smart contract vulnerability detection," in Journal of physics: conference series, 2021, vol. 1820, no. 1: IOP Publishing, p. 012004.

[43] L. S. H. Colin, P. M. Mohan, J. Pan, and P. L. K. Keong, "An integrated smart contract vulnerability detection tool using multi-layer perceptron on real-time solidity smart contracts," IEEE Access, vol. 12, pp. 23549-23567, 2024.

[44] Z. Wang, J. Chen, Y. Wang, Y. Zhang, W. Zhang, and Z. Zheng, "Efficiently detecting reentrancy vulnerabilities in complex smart contracts," Proceedings of the ACM on Software Engineering, vol. 1, no. FSE, pp. 161-181, 2024.

[45] Z. Zhen, X. Zhao, J. Zhang, Y. Wang, and H. Chen, "DA-GNN: A smart contract vulnerability detection method based on Dual Attention Graph Neural Network," Computer Networks, vol. 242, p. 110238, 2024.

[46] L. Zhang et al., "A novel smart contract reentrancy vulnerability detection model based on BiGAS," Journal of Signal Processing Systems, vol. 96, no. 3, pp. 215-237, 2024.

[47] H. Wu, H. Dong, Y. He, and Q. Duan, "Smart contract vulnerability detection based on hybrid attention mechanism model," Applied Sciences, vol. 13, no. 2, p. 770, 2023.

[48] Y. Sun et al., "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1-13.