

Artificial Intelligence for Security Analysis of Smart Contracts: A Systematic Review

Lívia Carrera
Instituto Militar de Engenharia (IME)
Departamento de Sistemas e
Computação
Rio de Janeiro, RJ, Brasil
livia.emanuelle@ime.eb.br

Raquel C.G. Pinto
Instituto Militar de Engenharia (IME)
Departamento de Sistemas e
Computação
Rio de Janeiro, RJ, Brasil
raquel@ime.eb.br

Anderson Fernandes Pereira
dos Santos
Instituto Militar de Engenharia (IME)
Departamento de Sistemas e
Computação
Rio de Janeiro, RJ, Brasil
Venturus Innovation Center
Campinas, SP, Brasil
anderson@ime.eb.br

ABSTRACT

This Systematic Literature Review (SLR) investigates the application of Artificial Intelligence (AI) techniques for smart contract vulnerability detection in Web3 decentralized applications (dApps). Smart contracts enable trustless finance, decentralized governance, and autonomous economic protocols; however, their immutability makes vulnerabilities deployed on-chain difficult to patch without costly governance interventions. Between 2020 and 2024, 127 high-impact attacks caused \$2.3 billion in cumulative losses, while state-of-the-art automated tools could have prevented only 8% of them (10 of 127, \$149M). Following PRISMA guidelines, records retrieved from IEEE Xplore, the ACM Digital Library, and Springer Link were screened, resulting in 93 primary studies selected for in-depth analysis. The review organizes the literature around three generations of AI-based detection: classical machine learning, deep learning and Graph Neural Networks (GNNs), and Large Language Models (LLMs). The findings show that LLM-based detectors often achieve high recall but limited precision, while GNNs obtain strong results on narrowly scoped economic attacks, such as sandwiching, but still struggle to generalize. Logic-layer and protocol-layer vulnerabilities, which account for a significant share of financial losses, remain insufficiently addressed. The review also highlights emerging directions, including cross-contract reasoning, automated repair, and the security implications of the post-quantum cryptography transition. Persistent challenges include dataset scarcity, limited explainability, weak generalization, and the lack of robust methods for complex cross-contract and protocol-level vulnerabilities. These insights provide guidance for future research toward more accurate, explainable, and comprehensive AI-based security analysis of smart contracts.

KEYWORDS

Smart Contracts, Web3 Security, Vulnerability Detection, Artificial Intelligence, Graph Neural Networks, Large Language Models, Post-quantum Cryptography, Systematic Review

1 INTRODUCTION

Web3 decentralized applications (dApps) execute application logic autonomously on distributed blockchain infrastructure, without centralized servers or trusted intermediaries [4]. A graphical overview of the smart-contract lifecycle that motivates this review is given in Figure 1. At their core are *smart contracts*: self-executing programs that enforce contractual terms automatically when predefined conditions are met [34]. With the maturation of Ethereum [4, 35], smart contracts enabled a rich ecosystem of decentralized finance (DeFi), non-fungible token (NFT) markets, decentralized autonomous organizations (DAOs), and cross-chain bridges, and blockchain is increasingly adopted across mainstream trade and finance systems [37].

The economic gravity of this ecosystem is substantial: total value locked (TVL) in DeFi exceeded \$180 billion at peak, and industry trackers estimate cumulative losses to smart-contract exploits on the order of tens of billions of dollars [19]. Yet adoption produced a *security paradox*. The same immutability that guarantees trustless execution means that on-chain vulnerabilities cannot be patched without costly governance interventions or full contract migrations [41]. Chaliasos et al. [5] catalogued 127 high-impact attacks causing \$2.3 billion in cumulative losses between 2020 and 2024. Strikingly, their evaluation of five state-of-the-art automated tools against those attacks found that the tools could have prevented only 8% (10 of 127, totalling \$149M)—and that every preventable case was reentrancy-related. Logic-level and protocol-level bugs, which dominate modern DeFi exploits, remain largely beyond current automated tooling.

Existing surveys establish taxonomies of vulnerabilities, analysis techniques, and empirical attack patterns [3, 14, 31, 41], classify hundreds of mitigation solutions [16], and compare detection techniques empirically [12, 30]. A second, AI-specific wave has recently appeared: Yang et al. [39] systematize 27 AI-powered analysis studies and benchmark three of their tools, and Alsunaidi et al. [1] survey 108 machine-learning detection methods with a fine-grained taxonomy. These works are comprehensive on *tooling and learning architectures*, but they neither quantify the distance between benchmark performance and real-world exploit prevention nor consider the post-quantum threat that will affect all deployed systems. This review fills that gap by aligning the three generations of AI detection with the empirically dominant loss classes and with the

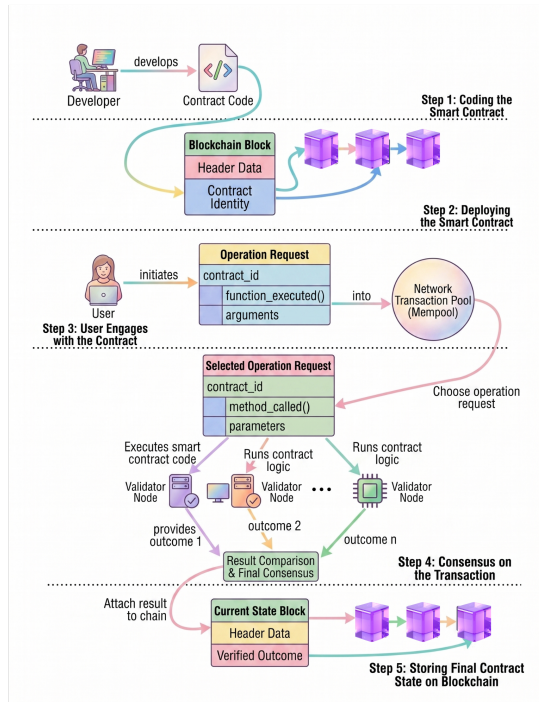


Figure 1: Graphical overview of the smart contract lifecycle, from development and deployment to transaction execution, validator consensus, and persistent blockchain state.

emerging cryptographic transition. The study is driven by three research questions—spanning AI techniques, residual vulnerability classes, and emerging post-quantum threats—which we formalize together with the review protocol in Section 2.

This survey contributes: (C1) a PRISMA-based systematic review of AI for Web3 security, with a reproducible PICOC protocol; (C2) a risk-oriented taxonomy of dApp vulnerability surfaces; (C3) a comparative analysis of three generations of AI detection methods, updated with the most recent GNN, LLM, cross-contract, and automated-repair evidence; (C4) a critical assessment of the practitioner gap; and (C5) a consolidated research agenda. Section 2 details the methodology and research questions; Section 3 presents the vulnerability taxonomy; Sections 4 and 5 analyze traditional and AI-based techniques; Section 6 discusses emerging threats; and Section 7 discusses the findings and consolidates the conclusion.

2 METHODOLOGY

This study was conducted as a systematic literature review (SLR), a structured and reproducible method for identifying, classifying, and synthesizing the evidence on a research topic [17]. To standardize planning, conduct, and reporting, we adopted the PRISMA 2020 statement [27] and managed the whole protocol—research questions, PICOC formulation, search strings, selection criteria, quality assessment, and data extraction—with the *Parsifal* tool [28], an online platform for collaborative SLRs that records the transition

Table 1: PICOC protocol and the keywords feeding the search string.

Criterion	Definition	Keywords
Population	Smart contracts / Web3 dApps	“smart contract”, “decentralized application”, “Web3”
Intervention	AI-based security analysis	“artificial intelligence”, “machine learning”, “deep learning”, “GNN”/“graph neural network”, “LLM”/“large language model”
Comparison	Traditional analysis (auxiliary)	“static analysis”, “symbolic execution”, “formal verification”, “fuzzing”
Outcome	Vulnerability detection	“security”, “vulnerability”, “detection”, “analysis”, “audit”
Context	Peer-reviewed + standards	IEEE, ACM, Springer; NIST, Ethereum Foundation (no query terms)

of each study through the selection stages. The methodology is organized into the following stages: research questions (2.1), PICOC formulation (2.2), data sources (2.3), search strategy (2.4), selection criteria (2.5), quality assessment (2.6), data extraction (2.7), and study selection (??); it closes with the data-extraction and quality-assessment summary (2.8), explicit reproducibility (2.9), and threats-to-validity analyses (2.10), so that every selection and reporting decision is auditable.

2.1 Research Questions

The review is driven by three research questions whose shared subject (its *population*) is the security of smart contracts in Web3 decentralized applications:

- RQ1.** What AI-based techniques have been proposed for smart contract vulnerability detection, and what are their relative strengths, limitations, and detection coverage?
- RQ2.** What vulnerability classes in smart contracts remain inadequately addressed by current AI approaches, and what are the open research challenges?
- RQ3.** What emerging threats—including the post-quantum cryptography transition—will shape the next decade of Web3 security?

2.2 PICOC and Search-String Derivation

To make the questions operational and to derive the search string systematically rather than ad hoc, we structured the protocol with the PICOC criteria (Population, Intervention, Comparison, Outcome, Context) [17], summarized in Table 1. The **Population** is the unit common to all three research questions—smart contracts and Web3 dApps—which is why it anchors every query. The **Intervention** is the family of AI-based detection techniques (RQ1); the **Comparison** is the body of traditional techniques against which they are assessed; the **Outcome** is security/vulnerability detection, analysis, and audit (RQ2); and the **Context** is peer-reviewed scientific literature (IEEE Xplore, the ACM Digital Library, and Springer Link) complemented by authoritative technical standards—NIST [26] and the Ethereum Foundation [10]—which are also primary sources for the emerging-threats analysis (RQ3).

Table 2: Relation between research questions and search-string fragments.

RQ	Focus	PICOC terms
RQ1	AI detection techniques	Population $\wedge$ Intervention $\wedge$ Outcome
RQ2	Residual vulnerability classes	Population $\wedge$ Outcome ("vulnerability", logic/protocol)
RQ3	Post-quantum / emerging threats	Population $\wedge$ ("quantum" $\vee$ "post-quantum")

The Population, Intervention, and Outcome criteria compose the primary query; the Comparison criterion supplies auxiliary terms used to retrieve baseline studies on traditional techniques. Table 2 makes explicit how each research question maps onto the PICOC terms that retrieve its evidence, so that the link between questions and search is auditable.

2.3 Data Sources and Digital Libraries

Guided by the Context criterion, we queried the digital libraries with the broadest coverage of the computer-security and AI venues relevant to smart contracts. The libraries were selected against four criteria: *relevance* (they publish the original peer-reviewed versions); *query expressiveness* (fielded Boolean search with date filtering, which makes the canonical string reproducible); *full-text accessibility* (required for eligibility and quality assessment); and *coverage breadth* across publishers. Accordingly, **IEEE Xplore** and the **ACM Digital Library** were used as the primary sources of original research, and **Springer Link** was added as a broad, cross-publisher index; other multidisciplinary aggregators (e.g., Scopus and Web of Science) were not queried separately because they predominantly re-index these same venues and would not add unique in-scope primary studies. We complemented the libraries with the technical standards of NIST [26] and the Ethereum Foundation [10] as primary sources for the post-quantum material.

2.4 Search Strategy

The keywords were organized around the PICOC concepts and combined with the Boolean operators AND/OR, so that Parsifal could generate systematic and reproducible strings. The canonical string was *adapted to the syntax and indexing of each database*—each imposes its own field syntax, indexing, and date-filter restrictions—so it was *not* executed verbatim. We therefore report the single, database-agnostic canonical string and note the per-database adaptations rather than reproducing one variant per repository. The canonical (general) string was:

```
("smart contract" OR "decentralized application" OR "Web3")
AND ("security" OR "vulnerability")
AND ("artificial intelligence" OR "machine learning"
OR "deep learning" OR "graph neural network"
OR "large language model")
AND ("detection" OR "analysis" OR "audit")
```

Auxiliary queries addressed post-quantum cryptography in blockchain ("post-quantum" AND "blockchain", feeding RQ3), formal verification, and DeFi-specific attacks, to retrieve baseline and emerging-threat evidence not captured by the primary string.

2.5 Selection Criteria

The selection of studies was based on the predefined inclusion and exclusion criteria below.

Inclusion criteria.

- Articles written in English or Portuguese, available in full text.
- Peer-reviewed studies or recognized technical standards.
- Studies that propose or evaluate AI-based (ML, DL, GNN, or LLM) techniques for smart contract security, or that provide empirical analysis of real-world attacks.
- Articles published between 2020 and 2026, plus foundational reference works of the field regardless of period.

Exclusion criteria.

- Studies that do not address smart contract security.
- Duplicate records across databases.
- Studies lacking sufficient methodological detail (quality score below the cut-off, Section 2.6).
- Works focused exclusively on price prediction or other non-security topics.

2.6 Quality Assessment Checklist

To ensure methodological rigor, each candidate study retrieved for full-text assessment was evaluated with a five-question Quality Assessment Checklist covering relevance, transparency, and methodological soundness:

- (1) Does the study propose or evaluate an AI-based technique (ML, DL, GNN, or LLM) for smart contract security analysis?
- (2) Is the dataset or benchmark used clearly described and, ideally, publicly accessible?
- (3) Does the study report quantitative detection metrics (e.g., precision, recall, F1-score, accuracy)?
- (4) Is the experimental methodology—baselines, evaluation protocol, and setup—described in enough detail to be reproducible?
- (5) Does the study present *and discuss* its results, including limitations?

Each question was scored as **Yes** (1.0), **Partially** (0.5), or **No** (0.0), for a maximum of 5.0 points. A cut-off of **3.0** points was adopted: only studies scoring at or above the threshold were retained for the synthesis, which operationalizes the "insufficient methodological detail" exclusion criterion.

2.7 Data Extraction Form

A structured data-extraction form was designed to collect comparable information from every included study. Table 3 summarizes the fields, which feed the comparative analyses in Sections 4 and 5.

Figure 2 presents the resulting PRISMA 2020 flow. In **identification**, 1,903 records were retrieved and 187 duplicates were removed before screening, leaving 1,716. The anchor totals are the 1,903 in-scope identified records (after scope-filtering the IEEE Xplore, ACM, and Springer), the 111 reports assessed in full text, and the 93 studies included. During **screening**, 1,605 records were excluded by title and abstract, and the remaining 111 reports were retrieved in full. At the **eligibility** stage, the 111 full texts were assessed and 18 were excluded—9 not addressing AI for smart-contract security, 5

Table 3: Data-extraction fields.

Field	Values
Platform	Ethereum/EVM, Solana, Aptos/Sui, StarkNet, other
Technique family	Static, symbolic, formal, fuzzing, classical ML, DL/GNN, LLM
AI generation	Gen 1 (classical ML), Gen 2 (DL/GNN), Gen 3 (LLM)
Vuln. classes	Reentrancy, access control, oracle, logic/protocol, among others
Input representation	Source, bytecode, CFG/DFG, transaction graph
Quantum aspects	Considered / not considered
Metrics	Precision, recall, F1, accuracy
Reproducibility	Artifact / dataset availability

below the quality cut-off, and 4 focused on non-security topics such as price prediction—so that **93 primary studies** were included. These studies are catalogued as P1–P93 in Appendix A; the corresponding evidence is synthesized in the PRISMA and temporal figures and in the categorization tables, while the works discussed in depth—together with methodological, normative, and foundational sources—appear in the reference list [4, 10, 17, 25–27, 34, 35]. Figure 3 shows the temporal distribution of the included studies, which concentrates in 2024–2026 and reflects the recent surge of LLM- and GNN-based detection work. Although the search window opened in 2020 (Section 2.5), the earliest *included* study dates from 2021; no 2020 record satisfied the inclusion and quality criteria, so the distribution begins at 2021.

2.8 Data Extraction and Quality Assessment Summary

For each included study we extracted the fields of Table 3: publication year and venue; blockchain platform targeted; security analysis technique and AI generation; vulnerability types addressed; input representation; quantum-computing considerations; reported detection metrics; and reproducibility of results. Each paper was additionally scored with the checklist of Section 2.6; studies below the 3.0 cut-off were excluded from the primary synthesis but retained as contextual references where relevant. Official documentation from NIST [26] and the Ethereum Foundation [10] was incorporated as primary source material for the post-quantum analysis.

2.9 Reproducibility and Protocol

To minimize researcher degrees of freedom, the protocol was specified *before* execution: the research questions, the PICOC mapping (Table 1), the canonical search string and its per-database adaptations (Section 2.4), the inclusion and exclusion criteria (Section 2.5), the five-item quality checklist with its 3.0 cut-off (Section 2.6), and the data-extraction form (Table 3) were frozen prior to screening and recorded in Parsifal, which logs every state transition of each record. Selection is therefore *deterministic* given these artifacts: a record is included if and only if it satisfies every inclusion criterion, violates no exclusion criterion, and scores at least 3.0 on the quality checklist. The complete screening sheet—identifiers, decisions, venues, years, and Digital Object Identifiers (DOIs) of all 111 assessed reports and the 93 included studies—underlies the P1–P93

catalogue of Appendix A and the coding used in Tables 6–8, so that the funnel of Figure 2 and the synthesized categories can be reproduced end to end. To support reproducibility and facilitate further research, the complete replication package of this systematic literature review is publicly available on GitHub. The repository contains the datasets, study-selection records, classification artifacts, extracted data, and supporting material used in the review. The replication package is available at:

<https://github.com/Lixipluv/RSL-Smart-Contract>

2.10 Limitations and Scope

The validity of this review is subject to some limitations. Selection bias was mitigated through a structured PRISMA protocol applied across title, abstract, and full-text screening, although the absence of inter-rater evaluation remains a limitation. The classification of the 93 studies by their dominant approach, algorithm, and metric (Tables 6–8) may underrepresent secondary techniques in hybrid systems. Moreover, the proposed three-generation framework simplifies a partly overlapping research landscape, and the risk taxonomy relies on a single empirical dataset [5]. Terminological inconsistencies across studies may also have affected retrieval, partially mitigated through synonym-based auxiliary queries [17].

3 VULNERABILITY TAXONOMY

We propose a risk-oriented taxonomy of dApp vulnerability surfaces organized into five tiers (Table 4), grounded in the risk-management framework of ISO 31000 [15] and derived from Chaliasos et al.’s analysis of 127 attacks [5] and complementary DeFi-incident data [42]. ISO 31000 defines risk as the effect of uncertainty on objectives, assessed through the combination of an event’s *likelihood* and its *consequence*. We adopt this principle to order the tiers by *realized risk*—the product of how frequently a surface is attacked and the magnitude of the resulting loss—rather than by syntactic severity. This is why a syntactically simple but high-consequence and frequently exploited surface (e.g., DeFi oracle and price-manipulation logic) outranks a more intricate but rarely monetized one: aligning the taxonomy with the consequence-weighted view of risk that ISO 31000 prescribes lets practitioners prioritize treatment where expected loss, not code complexity, is greatest. **Tier 1** (DeFi, lending/AMM, bridges, upgradeable protocols) concentrates 68% of attacks and roughly \$1.6 billion in losses; within it, cross-chain bridges produce few but extremely costly incidents, averaging more than \$350M per event (Poly Network \$611M, Ronin \$625M, Wormhole \$326M). **Tier 2** covers NFTs, token standards, and stablecoins; **Tier 3** governance and oracle systems; **Tier 4** real-world-asset tokenization and payments; and **Tier 5** privacy and parametric-insurance protocols. Two observations guide the rest of the paper. First, oracle manipulation recurs across multiple tiers, indicating a cross-cutting concern rather than an isolated bug. Second, several Tier 1 vectors—storage collision (a direct consequence of upgradeable proxies [22]), flash-loan-enabled economic manipulation, and MEV/sandwich attacks—are *logic-* or *protocol-level* rather than purely syntactic, which is precisely where current detection underperforms.

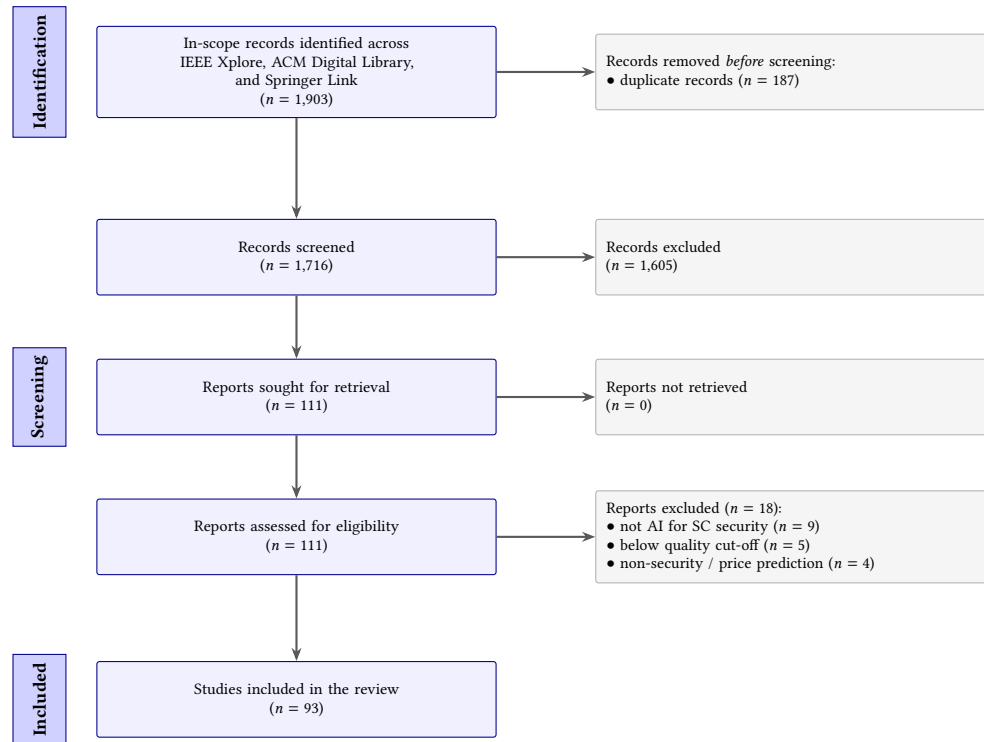


Figure 2: PRISMA 2020 flow diagram of the identification of studies via databases and registers.

Table 4: Risk-oriented taxonomy of smart contract applications (Tier 1 concentrates 68% of attacks and ~\$1.6B in losses).

Tier	Applications	Attack Vectors
T1 – High 68% of attacks	DeFi / DEX Lending / AMM Bridges Upgradeable protocols	Flash loans Oracle manipulation MEV / Sandwich Signature replay Storage collision
T2 – Med-High	NFTs Token standards Stablecoins	Reentrancy via callbacks Depeg attacks Approve race condition
T3 – Medium	DAOs Oracles Identity	Flash loan governance Oracle front-running Revocation failure
T4 – Low-Med	RWA tokenization Payments Social tokens	Oracle dependency Timestamp manipulation Rug pulls
T5 – Special	Privacy protocols Parametric insurance	ZK-proof exploits Nullifier reuse Oracle manipulation

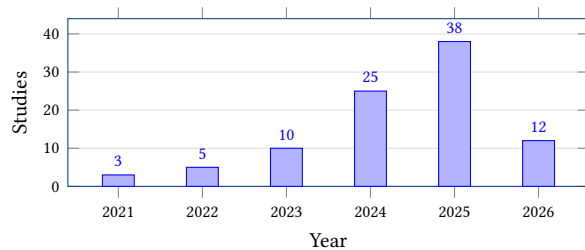


Figure 3: Distribution of the 93 included primary studies by publication year (2021–2026).

3.1 Tier 1 in Depth

Because Tier 1 concentrates the losses, it warrants closer treatment. **DeFi** decomposes into decentralized exchanges and AMMs (exposed to front-running, MEV, sandwich attacks, and price-impact manipulation), lending protocols (where flash loans enable atomic borrow–manipulate–repay exploits), yield/staking (reentrancy, share dilution), and synthetic-asset platforms (funding-rate and oracle manipulation). Sandwich attacks are emblematic of the protocol layer: they occur roughly once or twice per block by bracketing a pending user trade with attacker buy/sell transactions, and increasingly hide in the mempool or span multiple blocks, evading rigid heuristic detectors [36]. **Cross-chain bridges** lock assets on a source chain and mint representations on a destination

chain; although few in number, they exhibit the highest average loss per incident, with the Poly Network, Ronin, and Wormhole exploits each exceeding \$300M and stemming from signature-replay, validator-set compromise, or insufficient proof verification.

Upgradeable proxies—Transparent, UUPS (EIP-1822), and Diamond (EIP-2535)—restore patchability but introduce storage collisions, uninitialized-implementation hijacking, function-selector collisions, and a single high-value admin key whose compromise grants control of all funds. These admin keys also become permanent post-quantum targets (Section 6).

3.2 Vulnerability Classes by Layer

Orthogonal to the application tiers, smart contract vulnerabilities can also be understood according to the layer in which they emerge [3, 31]. At the **Solidity source-code layer**, common vulnerabilities include reentrancy, integer overflow and underflow, access-control errors, unchecked external calls, mishandled exceptions, and denial-of-service conditions caused by unexpected reverts. Reentrancy occurs when an external call is performed before the contract state is updated, allowing the callee to re-enter the contract, as illustrated by the 2016 DAO attack, which drained approximately 3.6M ETH. Integer overflow and underflow vulnerabilities have been largely mitigated since Solidity $\geq 0.8.0$ introduced checked arithmetic by default, whereas access-control flaws, such as missing `onlyOwner` modifiers or unsafe `tx.origin` authentication, remain recurring sources of risk. More challenging than these syntactic bugs are *logical flaws*, in which the implementation is valid from a programming perspective but violates the intended protocol invariants.

At the **Ethereum Virtual Machine (EVM) bytecode layer**, vulnerabilities include short-address attacks, call-stack-depth abuse, and unsafe `delegatecall` usage. The 2017 Parity incident, in which a library self-destruct froze roughly \$150M, illustrates how low-level execution semantics can have severe financial consequences. Finally, at the **block-dependency layer**, vulnerabilities arise from assumptions about blockchain state and transaction ordering. These include timestamp and block-hash dependence, transaction-ordering dependence, maximal extractable value (MEV), and oracle manipulation, which underpins a large share of flash-loan-based exploits.

This layered view helps explain why current automated detectors remain limited in practice. The vulnerability classes responsible for the largest losses are often not simple source-code patterns, but logic-level, protocol-level, and economic flaws that depend on contract interactions, market assumptions, or runtime context. As a result, they lack the clear syntactic signatures on which many detection tools rely, helping to explain the reported 8% prevention rate and motivating the analysis of detection techniques that follows.

4 TRADITIONAL ANALYSIS TECHNIQUES

Smart-contract analysis predates AI-based detection and is conventionally organized into *four* families—static analysis, symbolic execution, formal verification, and fuzzing—together with the hybrid pipelines that compose them [3, 30, 38]. These families are not at the same conceptual level: they differ in the input they consume

(source, bytecode, or runtime state), in whether they execute the contract, and in the guarantees they offer (heuristic versus sound). Figure 4 organizes them as a taxonomy rather than a flat list, before each family is discussed in turn.

Table 5 summarizes their inputs, strengths, limitations, and practitioner adoption. **Static analysis** (Slither, Securify, Vandal) detects known patterns through taint and control-flow analysis. Slither is the most adopted tool in practitioner workflows [5] thanks to speed and a rich detector library, but it cannot reason about logic-layer behaviour. **Symbolic execution** (Oyente [21], Mythril, Manticore) invokes SMT solvers to find vulnerability-triggering inputs; path explosion is its fundamental scalability barrier. **Formal verification** provides soundness guarantees: the Move Prover [9] verifies the entire Diem framework in seconds-to-minutes, demonstrating feasibility for production code, while property-based testing in ConCert [23] and Coloured-Petri-Net models [13] target specific classes such as reentrancy at the cost of expert specification effort. **Fuzzing** explores executions dynamically; the recent profit-centric fuzzer VERITE [18] aligns search with attacker incentives, detecting 29 of 61 real-world exploits (versus 10 for ItyFuzz), extracting **134× more profit on average**, and in 12 cases exceeding the profit realized by the actual attackers; deployed in production it surfaced six zero-day vulnerabilities (five high-severity). Despite this progress, the empirical 8% prevention rate [5] establishes the practitioner gap that AI-based approaches must close.

4.1 Static Analysis

Static analysis inspects source or bytecode without execution, detecting known patterns via taint analysis, control-flow analysis, and abstract interpretation. Its advantages are speed and the absence of false negatives for the encoded rules; its limits are false positives and blindness to logic-layer bugs without syntactic signatures. Slither operates over an intermediate representation built from the abstract syntax tree (AST) and ships more than 70 detectors covering reentrancy, access control, and uninitialized variables; Securify formalizes *compliance* and *violation* patterns over symbolic data flow, enabling sound classification rather than heuristics; Vandal and Gigahorse lift EVM bytecode to a declarative or three-address form amenable to Datalog analysis, important because the majority of deployed contracts publish no source.

4.2 Symbolic Execution

Symbolic execution treats inputs as symbolic values and solves the accumulated path constraints with a satisfiability-modulo-theories (SMT) solver (Z3, CVC5) to synthesize vulnerability-triggering inputs. Oyente [21], the first such tool for Ethereum, identified transaction-ordering, timestamp, mishandled-exception, and reentrancy bugs, and was evaluated on 19,366 contracts—including many of the highest-balance contracts of the time. Mythril adds taint tracking, Manticore exposes a scriptable Python API for custom analyses, and MAIAN targets *trace properties* across multiple invocations (prodigal, suicidal, greedy contracts). The shared limitation is path explosion as control-flow complexity grows.

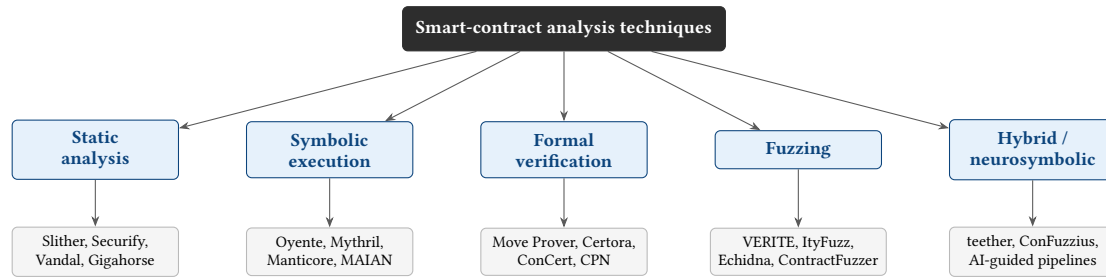


Figure 4: Taxonomy of smart-contract analysis by techniques.

Table 5: Traditional smart contract analysis techniques.

Technique	Input	Strengths	Limitations	Adoption
Static analysis Slither, Securify, Vandal	Bytecode Source code	✓ Fast, scalable ✓ Strong on canonical bugs	× Misses logic-layer bugs × Pattern-bound	High
Symbolic exec. Oyente, Mythril, Manticore	Bytecode Source code	✓ Finds triggering inputs ✓ Path-precise	× Path explosion × Poor scalability	Medium
Formal verif. Move Prover, ConCert, CPN	Source code	✓ Soundness guarantees ✓ Production-grade speed	× Manual specifications × Narrow scope	Low
Fuzzing VERITE, ItyFuzz, Echidna	Runtime	✓ Profit-centric search ✓ 134× profit vs ItyFuzz	× Coverage-bound × No formal guarantee	Medium

4.3 Formal Verification

Formal verification yields mathematical guarantees relative to a specification. The Move Prover [9] stands out: developers write pre/post conditions and global resource invariants inline in the Move Specification Language, and the prover discharges them with SMT-based deductive reasoning, verifying the entire Diem framework in seconds-to-minutes—fast enough for continuous integration. The Certora Prover brings comparable specification-driven verification to Solidity (CVL), at significant specification cost, and is used by major DeFi protocols. Lighter-weight property-based and model-based approaches such as ConCert [23] and Coloured-Petri-Net reentrancy models [13] target specific classes with less up-front effort.

4.4 Fuzzing and Hybrid Approaches

Fuzzing explores dynamic executions. Echidna falsifies user-specified properties; ContractFuzzer instruments the EVM for black-box runtime checks; and ItyFuzz introduced snapshot-based fuzzing for stateful DeFi protocols. The profit-centric VERITE [18] reorients the objective from generic crash discovery to *economically exploitable* bugs through three innovations: DeFi action-based mutators (e.g., swap, deposit, and borrow), profitability criteria over abnormal fund flows, and gradient-descent profit maximization. On 61 real-world exploited projects it detected 29 vulnerabilities to ItyFuzz’s 10 and extracted 134× more profit on average. Finally, *hybrid* pipelines compose techniques: static analysis narrows the search space for symbolic execution (teether), symbolic execution seeds fuzzers (ConFuzzius), and concolic execution balances scalability with path precision—foreshadowing the neurosymbolic AI methods of Section 5.

5 AI-BASED TECHNIQUES

We organize AI-based detection into three generations (Table 9), tracing a progression from feature-engineered classifiers to structure-aware deep models and, most recently, to language models with contextual reasoning. To characterize the whole corpus rather than only the systems discussed in depth, Tables 6–8 categorize all 93 included primary studies—coded as P1–P93 during extraction—by detection approach, learning algorithm, and reported evaluation metric. The distribution confirms three trends: graph- and LLM-based detection dominate the recent literature, surveys and empirical studies form a sizeable methodological backdrop, and accuracy and F1 remain the most frequently reported metrics. Following established taxonomies [1, 3, 30], vulnerabilities manifest at three layers—the Solidity (source) layer, the EVM (bytecode) layer, and the block-dependency layer—and each AI generation targets these layers with different representations. Several characteristics make the problem amenable to machine learning: hundreds of thousands of contracts are publicly available with tens of thousands documented as vulnerable; many bugs exhibit learnable structural patterns; code admits natural sequence and graph representations; and the financial stakes justify substantial computational investment. This section answers RQ1 and motivates the limitations discussed under RQ2.

5.1 Generation 1: Classical Machine Learning

Early approaches applied decision trees, random forests, and SVMs to features extracted from contract code or bytecode—opcode-frequency distributions, control-flow-graph metrics, and N-gram features. Momeni et al. [24] adapted two static analyzers to label more than 1,000 verified Ethereum contracts and trained classifiers reaching an average accuracy of 95%, showing that static features

Table 6: Detection approaches across the primary studies (each study is classified by its main contribution).

Approach	Studies	Count
Graph-based detection (GNN)	[2, 4, 6, 7, 9, 14, 17, 24, 26, 28, 32, 33, 41, 51, 52, 55, 60, 65, 68, 70, 71, 83, 84]	23
LLM-assisted auditing	[12, 15, 16, 23, 34, 45, 54, 61, 62, 64, 67, 75, 76, 77, 81, 85, 88, 89, 90, 91, 92]	21
Surveys and empirical studies	[13, 20, 21, 25, 29, 39, 40, 42, 44, 48, 49, 50, 59, 63, 74, 82, 86, 93]	18
Sequence models (LSTM/RNN/CNN)	[5, 8, 18, 37, 38, 56, 73, 80, 87]	9
Fuzzing and dynamic analysis	[19, 27, 47, 53, 69, 72]	6
Code/comment generation	[43, 46, 78, 79]	4
Static/symbolic analysis	[3, 22, 31]	3
Multimodal feature fusion	[1, 10, 11]	3
Pretrained transformers	[30, 58, 66]	3
Automated repair	[35, 36]	2
Other deep networks	[57]	1

Each bracketed number n denotes primary study Pn (e.g., 2 = P2) in the catalogue of Appendix A.

Table 7: Predominant learning algorithms across the primary studies.

Algorithm	Studies	Count
Large language models	[12, 15, 16, 20, 23, 29, 34, 39, 40, 45, 46, 49, 54, 59, 61, 62, 63, 64, 67, 75, 76, 77, 78, 79, 81, 85, 88, 89, 90, 91, 92]	31
Graph neural networks	[2, 4, 6, 7, 9, 14, 24, 26, 28, 32, 33, 41, 51, 52, 60, 65, 68, 69, 70, 71, 74, 83, 84]	23
Pretrained transformers (BERT/DeBERTa)	[1, 3, 10, 11, 21, 27, 30, 35, 50, 53, 58, 66, 72, 82, 86, 93]	16
Others (static, symbolic, ensembles)	[17, 19, 22, 25, 31, 42, 43, 47, 48, 55, 57]	11
LSTM/BiLSTM/RNN	[5, 37, 38, 56, 73, 80, 87]	7
Transformers (generic)	[8, 13, 44]	3
CNN/TextCNN	[18]	1
Reinforcement learning	[36]	1

Each bracketed number n denotes primary study Pn (e.g., 2 = P2) in Appendix A. Classified by each study's core algorithm; "Others" aggregates static, symbolic, and ensemble approaches.

alone suffice for statistical detection of common vulnerabilities. In the closely related fraud-detection setting, Chen et al. [7] combined account features with opcode frequencies to identify smart Ponzi schemes directly from on-chain data, estimating more than 500 active schemes on Ethereum. This fraud-detection line illustrates that on-chain *data mining* predates the AI techniques studied here: scam detection has been pursued both with learned classifiers and with non-AI heuristics and address-clustering over transaction graphs, a baseline that contextualizes the value AI adds. Sequence- and image-based classical variants (e.g., S-gram and Color-CNN) handle reentrancy and integer overflow reasonably well but struggle with multi-step attacks, because hand-built features discard

Table 8: Evaluation metrics reported across the primary studies (a study may report several).

Metric	Studies	Count
Accuracy	[1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 21, 24, 27, 28, 31, 32, 33, 37, 39, 43, 44, 45, 46, 47, 48, 50, 51, 52, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 73, 74, 75, 76, 77, 78, 80, 83, 85, 87, 88, 90, 93]	60
F1-score	[1, 2, 4, 5, 6, 7, 8, 9, 14, 16, 18, 19, 22, 24, 26, 29, 30, 35, 37, 38, 42, 43, 44, 49, 51, 52, 54, 61, 62, 65, 66, 67, 68, 69, 70, 71, 73, 74, 75, 80, 81, 83, 84, 85, 88, 89, 90, 91, 92]	49
Precision / Recall	[1, 2, 5, 6, 7, 8, 9, 10, 12, 15, 18, 20, 22, 23, 24, 25, 28, 34, 35, 36, 40, 41, 43, 49, 52, 54, 57, 61, 65, 66, 67, 69, 70, 73, 74, 75, 81, 82, 83, 84, 85, 88, 90, 92, 93]	45
AUC / ROC	[2, 5, 14, 34, 35, 41, 49, 52, 53, 55, 56, 58, 65, 71, 72, 75, 79, 85]	18
FPR/TPR/FNR/TNR	[5, 15, 55, 86]	4

Each bracketed number n denotes primary study Pn (e.g., 2 = P2) in Appendix A. Multi-label count: the sum exceeds 93 because many studies report multiple metrics.

the program structure that distinguishes benign from malicious control flow.

5.2 Generation 2: Deep Learning and GNNs

Deep models reduce feature-engineering effort and learn richer representations. **Sequential models** based on LSTMs capture long-range opcode dependencies, but treat code as a flat token stream and miss graph structure. **Graph Neural Networks** address this directly: DR-GCN operates over control-flow and data-flow graphs and substantially outperforms rule-based tools on reentrancy, while AMEVulDetector adds attention over the abstract syntax tree and control-flow graph, supporting multi-type detection with partial explainability. Liu et al. [20] fuse interpretable graph features with expert patterns, narrowing the gap between learned representations and human-auditable rules. The graph paradigm is now maturing toward production-scale economic-attack detection and cross-contract reasoning, two fronts that single-contract classifiers could not reach. **Transaction-graph GNNs** target the protocol layer directly: SandWatch couples an order-independent heuristic (which prunes over 94% of transactions) with a dual-task GNN that jointly classifies sandwich-attack transactions and DEX nodes, trained under a positive-unlabeled strategy to exploit abundant unlabeled data; it reaches an F1 of 99.78% on a benchmark and, deployed over five months of 2024 Ethereum traffic, surfaced more than 560,000 sandwich attacks including mempool-only, cross-block, and pool-failure variants [36]. **Cross-contract GNNs** attack the composition problem: CrossMiner builds a cross-contract dependency graph from function-call chains and applies a heterogeneous GNN with a two-level attention mechanism, improving detection accuracy and F1 by roughly five percentage points over single-contract baselines on reentrancy, timestamp dependency, and transaction-state dependency [19]. **Pretrained transformers** such as SmartBERT [40] bring transfer learning and semantic understanding to Solidity, at a high pre-training cost and with a hard dependency on available source code. **Hybrid bytecode methods** such as ReChecker pair

symbolic execution with ML over bytecode, an important property given that over 77% of deployed Ethereum contracts lack public source code [41]—the same gap that motivated Oyente’s original bytecode evaluation on 19,366 contracts [21].

5.3 Generation 3: Large Language Models

LLMs introduce contextual reasoning and natural-language explanation. **GPTScan** [32] applies GPT-4 through structured prompting to flag logic vulnerabilities, achieving competitive recall but with significant hallucination risk and no formal guarantee. Empirical scrutiny tempers the enthusiasm: in a controlled study over the SmartBugs-curated dataset (142 contracts, nine DASP10 categories), Chen et al. [6] found that ChatGPT attains high recall but limited precision, that its performance varies sharply by vulnerability type, and that against 14 state-of-the-art tools its F-score is lower for three of seven vulnerabilities and only marginally higher for the rest—with answer uncertainty and input-length limits identified as core robustness barriers. **Hybrid LLM+static pipelines** (denoted **AuditAI** in Table 9) ground an LLM with the findings of a static analyzer: the static tool supplies ground-truth evidence while the LLM synthesizes audit-ready, human-readable reports, mitigating hallucination while retaining contextual reasoning. The maturation of open-weight models [8] is making such pipelines economically viable beyond proprietary APIs, and conversational-auditing workflows enable exploratory analysis that goes beyond predefined rules.

5.4 From Detection to Repair

A complementary thread moves beyond *flagging* bugs to *fixing* them automatically at the source level, which—given on-chain immutability—is most valuable before deployment. sGuard+ [11] guides rule-based repair with machine learning: it learns features that characterize both vulnerability symptoms and prevention idioms to cut false positives, localizes defects by traversing the abstract syntax tree, and refines the repair rules of its sGuard predecessor so that fixes preserve the original business logic with lower time and gas overhead. Automated repair is still narrow in scope and rule-bound, but it illustrates that the AI toolchain is broadening from pure detection toward an end-to-end secure-development loop.

5.5 Critical Limitations of AI Approaches (RQ2)

Chaliasos et al. [5] surveyed 49 developers and auditors, who used 4.5–5.3 different tool types on average and identified **logic-related bugs** and **protocol-layer vulnerabilities** as the most significant unaddressed threats. The 8% prevention rate reveals that the classes dominating financial losses—flash-loan exploits, oracle manipulation, cross-contract logic bugs, and MEV—resist syntactic pattern matching, and AI detectors inherit this difficulty. Independent benchmarking reinforces the point: comparing AI-powered tools (xFuzz, DLVA, GURU) with traditional analyzers, Yang et al. [39] find that the end-to-end AI classifiers gain efficiency but lose precision, and the ChatGPT study reaches the analogous conclusion that high recall is bought at the cost of precision and robustness [6]. The challenge is compounded by the observation that a *vulnerable* contract is not necessarily an *exploited* one [29]: models optimized for

vulnerability presence may misalign with the subset of flaws that are economically exploitable. The recurring limitations across the surveyed AI methods are: **hallucination** (LLMs produce plausible but incorrect findings); **no formal guarantees** (no mathematical proofs of correctness); **dataset dependency** (the absence of standardized vulnerability datasets is the single largest obstacle, a gap also emphasized by the broader ML survey [1]); **explainability** (deep models behave as black boxes); the **bytecode gap** (most deployed contracts lack source [41]); **cross-contract reasoning** (single-contract analysis misses composition-layer bugs, only recently addressed by heterogeneous-graph models [19]); and **adversarial robustness** (detectors can be evaded by adversarially crafted contracts). The most promising response is **neurosymbolic**: AI-guided symbolic execution, GNN representations augmenting expert rules [20], and LLM-generated specifications discharged by provers such as the Move Prover [9].

5.6 The Path Forward: Neurosymbolic Integration

The evidence suggests that neither pure AI nor pure formal methods will dominate; the most effective auditing will combine them. Four concrete patterns are emerging. *AI-guided symbolic execution* uses learned priors to rank the paths most likely to harbor vulnerabilities, focusing expensive solving on high-value regions and mitigating path explosion. *LLM+static pipelines* anchor language-model reasoning in static-analyzer ground truth, retaining contextual explanation while suppressing hallucination. *GNN+expert-rule* hybrids [20] fuse learned structural representations with hand-crafted detectors, capturing both well-understood and emergent patterns, and now extend to cross-contract graphs [19] and production-scale protocol-attack detection [36]. And *AI-based continuous monitoring* shifts defense from one-time pre-deployment audits to real-time anomaly detection over transaction flows, coupled with on-chain pause mechanisms and, increasingly, automated repair [11]. This direction simultaneously addresses the scalability of formal methods, the precision of AI, and the practitioner demand for explainable findings.

6 EMERGING THREATS AND FRONTIERS

The post-quantum cryptography (PQC) transition is an emerging threat that affects every deployed Web3 system at once [33]. At a high level it touches four layers of the Ethereum stack: account signatures (ECDSA), consensus attestations, data-availability commitments, and application-layer zero-knowledge proofs. We keep the focus on the account-signature layer, which has the most direct and immediate impact on smart contracts, and treat the remaining layers only at the level needed to convey scope. Ethereum accounts authenticate transactions with ECDSA over secp256k1. An address is a hash of the public key, so an account that has *never* transacted enjoys some protection; once it transacts, however, its public key is exposed, and Shor’s algorithm on a cryptographically relevant quantum computer (CRQC) recovers the private key in polynomial time—a one-way, permanent exposure. This enables a *harvest-now, decrypt-later* strategy in which adversaries record exposed public keys today and break them once a CRQC exists. The exposure is not uniform: long-lived, high-value targets such as multisignature

Table 9: Comparative analysis of AI-based smart contract security approaches.

Approach	Method	Input	Strengths	Limitations
<i>Generation 1 — Classical ML</i>				
S-gram / Color-CNN Gen 1	Classical ML	Bytecode	✓ Lightweight ✓ No source needed	× Limited expressiveness × Shallow patterns only
LSTM (opcode) Gen 1	RNN sequence	Bytecode	✓ Long-range opcode dep.	× Flat repr. × Misses graph structure
<i>Generation 2 — Deep Learning & GNNs</i>				
DR-GCN Gen 2	Graph Conv. Net	CFG/DFG	✓ High accuracy on reentrancy ✓ Structural awareness	× Source-code dependent × Complex graph construction
AMEVulDetector Gen 2	GNN + Attention	Source code	✓ Multi-type detection ✓ Partial explainability	× Scalability on large contracts
SandWatch Gen 2	Dual-task GNN + PU	Tx graph	✓ F1 99.78% on sandwich ✓ Mempool / cross-block variants	× Single attack family × Needs on-chain tx data
CrossMiner Gen 2	Heterog. GNN + Attn	Call-chain graph	✓ Cross-contract reasoning ✓ ~+5% acc./F1	× Few vuln. types × Graph construction cost
SmartBERT Gen 2	BERT fine-tuned	Source code	✓ Transfer learning ✓ Semantic understanding	× High pre-training cost × Source required
ReChecker Gen 2	Symbolic + ML	Bytecode	✓ No source needed ✓ Practical for deployed contracts	× Limited vulnerability scope
sGuard+ Gen 2	ML-guided repair	Source code	✓ Source-level auto-repair ✓ Preserves business logic	× Rule-bound classes × Detection → fix only
<i>Generation 3 — Large Language Models</i>				
GPTScan Gen 3	LLM (GPT-4)	Source code	✓ Contextual reasoning ✓ Explanatory output	× Hallucination risk × No formal guarantee
ChatGPT (empirical) Gen 3	LLM (GPT)	Source code	✓ High recall ✓ Zero-shot, no training	× Low precision × Uncertainty / code-length limits
AuditAI Gen 3	LLM + Static	Source code	✓ Mitigates hallucination ✓ Audit-ready reports	× Depends on static tool coverage

wallets, Ownable administrative accounts, upgradeable proxy admins, DAO treasuries, and time-locked vesting contracts—whose keys are reused over years—are disproportionately at risk.

NIST finalized its PQC standards in 2024 and set a migration timeline that drives the ecosystem’s response (Table 10), with ECDSA deprecation by 2030 and disallowance by 2035 [26]; Ethereum targets completion of core post-quantum infrastructure by approximately 2029 [10]. On the Ethereum side, EIP-8141 extends Account Abstraction with *signature agility*, letting each account migrate to a post-quantum scheme voluntarily, without breaking existing contracts. An open governance question—unresolved by the community—is how to handle *dormant wallets* whose owners may never migrate [10]. The same quantum advances also enable new consensus primitives such as proof of quantum work [2], underscoring that quantum computing is both a threat and an opportunity. The practical implication for this review is the need for **quantum-aware auditing tools** that flag long-lived contracts and bridges depending on obsolescent cryptography and assist in migration planning.

7 CONCLUSION AND FUTURE DIRECTIONS

This systematic review mapped the state of the art in AI-based smart contract security analysis. Following a PRISMA-compliant

Table 10: Compact NIST / Ethereum post-quantum migration timeline.

Year	Milestone
2024	NIST FIPS 203 (ML-KEM), 204 (ML-DSA), 205 (SLH-DSA) finalized
2030	ECDSA / classical signatures deprecated (NIST)
2035	ECDSA / classical signatures disallowed (NIST)
~2029	Ethereum core PQC infrastructure target; EIP-8141 (signature agility) as migration path

process, records retrieved from IEEE Xplore, the ACM Digital Library, and Springer Link were screened, resulting in 93 primary studies retained for analysis. The literature was organized around three generations of AI-based detection: feature-engineered machine learning, deep learning and Graph Neural Networks (GNNs), and Large Language Models (LLMs).

The evidence shows that AI-based detection has advanced substantially, especially through graph-based models and LLM-assisted auditing. GNNs have achieved strong results on specific economic attacks and are beginning to support cross-contract reasoning [19, 36], while LLM-based approaches provide broader semantic understanding but still face a recall–precision trade-off [6, 39]. However, these advances have not yet closed the gap between benchmark

performance and real-world exploit prevention. Empirical evidence shows that automated tools could have prevented only 8% of 127 real-world attacks, corresponding to 10 cases and \$149M out of \$2.3B in losses [5]. This practitioner gap is mainly explained by the fact that the most financially damaging vulnerabilities often occur at the logic, protocol, and economic layers, where syntactic detectors and narrowly scoped models remain limited.

The review also identified several directions for future research. First, the field needs standardized and multi-language benchmarks covering Solidity, Move, Cairo, logic-layer bugs, and DeFi-specific exploits [1]. Second, detection methods should become more profit-aware, aligning vulnerability discovery with the economic incentives of real attackers, as illustrated by profit-centric fuzzing approaches [18]. Third, neurosymbolic integration remains a promising path for combining the scalability of AI with the soundness of formal methods, including GNN-guided symbolic execution and LLM-generated specifications verified by formal provers [9, 20]. Fourth, cross-contract and composition analysis should be expanded, since many DeFi exploits emerge from interactions among multiple contracts rather than from isolated code fragments [19]. Fifth, automated repair and continuous monitoring should be further developed to support the full build-deploy-operate lifecycle [11, 36].

Finally, the post-quantum cryptography transition introduces a protocol-wide security concern that remains underexplored in current AI-based smart contract security research [33]. Future tools should therefore include quantum-aware auditing capabilities, such as detectors for long-lived cryptographic-obsolescence risks in contracts, bridges, and governance systems. Hybrid AI-quantum approaches, including quantum reservoir computing and related models, also represent a largely unexplored direction for addressing dataset scarcity and complex sequential attack patterns.

Overall, AI can significantly support smart contract auditing, but current methods remain insufficient for preventing the most damaging real-world attacks. Progress will depend on moving beyond isolated vulnerability classification toward risk-oriented, explainable, composition-aware, and quantum-aware security analysis. These directions define a research agenda for more accurate, practical, and resilient protection of Web3 systems.

8 ETHICAL CONSIDERATIONS

This study complies with Resolution No. 510/2016 of the Brazilian National Health Council (CNS). As a Systematic Literature Review, it did not involve direct research with human participants. Only scientific articles, technical documents, and publicly available bibliographic metadata were used. No personal data from participants were collected or analyzed, and bibliographic metadata were used solely to identify and cite the publications, in accordance with the applicable principles of the Brazilian General Data Protection Law (LGPD, Law No. 13,709/2018).

ACKNOWLEDGMENTS

The authors declare no conflicts of interest.

Declaration on the use of AI tools and technologies: Generative Artificial Intelligence tools, specifically ChatGPT (OpenAI) and Claude (Anthropic), were used within a limited and explicitly

defined scope: language refinement and proofreading of the manuscript, assistance with LaTeX formatting of tables and figures, and tabulation of the *already-decided* screening spreadsheet into the categorization tables and primary-studies catalogue. The systematic-review protocol itself (including the search, screening, eligibility assessment, quality assessment, and data extraction) was conducted by the authors, with reporting following the PRISMA 2020 guidelines. Generative AI tools were *not* used to make study-selection or eligibility decisions, assign quality scores, perform data extraction, or derive the analytical conclusions presented in this review.

The authors reviewed the complete manuscript and all supporting materials and take full responsibility for the content, analyses, and conclusions of this paper.

REFERENCES

- [1] S. J. Alsunaidi, H. Aljamaan, and M. Hammoudeh. 2025. Leveraging Machine Learning Models to Improve Smart Contract Security: A Survey of Vulnerabilities and Detection Methods. *Comput. Surveys* 58, 6 (2025), Article 151.
- [2] M. H. Amin, J. Raymond, D. Kinn, F. Hamze, K. Hamer, J. Pasvolosky, W. Bernoudy, A. D. King, and S. Kortas. 2025. Blockchain with Proof of Quantum Work. arXiv preprint.
- [3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts (SoK). In *Principles of Security and Trust (POST) (LNCS, Vol. 10204)*. Springer, 164–186.
- [4] Vitalik Buterin. 2014. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. <https://ethereum.org/en/whitepaper/>.
- [5] Stefanos Chaliasos, Marcos Antonios Charalambous, Liyi Zhou, Rafaila Galanopoulou, Arthur Gervais, Dimitris Mitropoulos, and Benjamin Livshits. 2024. Smart Contract and DeFi Security Tools: Do They Meet the Needs of Practitioners?. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*. ACM, New York, NY, USA, 1–13.
- [6] Chao Chen, Jiachi Su, Jintao Chen, Yanlin Wang, Ting Bi, Jing Yu, Yizhou Wang, Xin Lin, Ting Chen, and Zibin Zheng. 2025. When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We? *ACM Transactions on Software Engineering and Methodology* 34, 4 (2025), Article 100.
- [7] Weili Chen, Zibin Zheng, Edith Ngai, Peilin Zheng, and Yuren Zhou. 2019. Exploiting Blockchain Data to Detect Smart Ponzi Schemes on Ethereum. *IEEE Access* 7 (2019), 37575–37586.
- [8] DeepSeek-AI. 2024. DeepSeek LLM: Scaling Open-Source Language Models with Longtermism. arXiv preprint.
- [9] David L. Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Meng Xu, and Emma Zhong. 2022. Fast and Reliable Formal Verification of Smart Contracts with the Move Prover. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, Munich, Germany, 183–200.
- [10] Ethereum Foundation. 2024. Quantum Resistance. Accessed: 2024-11-15. <https://ethereum.org/roadmap/future-proofing/quantum-resistance/>
- [11] Chao Gao, W. Yang, J. Ye, Y. Xue, and J. Sun. 2024. sGuard+: Machine Learning Guided Rule-Based Automated Vulnerability Repair on Smart Contracts. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), Article 114.
- [12] W. Haouari, A. S. Hafid, and M. Fokaefs. 2024. Vulnerabilities of Smart Contracts and Mitigation Schemes: A Comprehensive Survey. arXiv preprint arXiv:2403.19805.
- [13] Y. He, H. Dong, H. Wu, and Q. Duan. 2023. Formal Analysis of Reentrancy Vulnerabilities in Smart Contract Based on CPN. *Electronics* 12, 10 (2023), 2152.
- [14] B. Hu, Z. Zhang, J. Liu, Y. Liu, J. Yin, R. Lu, and X. Lin. 2021. A Comprehensive Survey on Smart Contract Construction and Execution: Paradigms, Tools, and Systems. *Patterns* 2, 2 (2021), 100179.
- [15] International Organization for Standardization. 2018. ISO 31000:2018 — Risk Management: Guidelines. ISO, Geneva, Switzerland.
- [16] N. Ivanov, C. Li, Q. Yan, Z. Sun, Z. Cao, and X. Luo. 2023. Security Defense for Smart Contracts: A Comprehensive Survey. *Comput. Surveys* 55, 14s (2023), 1–37.
- [17] B. Kitchenham and S. Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE-2007-01. Keele University.
- [18] Ziqiao Kong, Cen Zhang, Maoyi Xie, Ming Hu, Yue Xue, Ye Liu, Haijun Wang, and Yang Liu. 2025. Smart Contract Fuzzing Towards Profitable Vulnerabilities. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 23.
- [19] X. Liu, T. Huang, C. Tan, J. Huang, Q. Wang, and Y. Pang. 2025. CrossMiner: Smart Contract Vulnerability Detection in Interactive Scenarios. In *Proc. IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, Piscataway, NJ, USA, 1724–1731.

- [20] Zhenguang Liu, Peng Qian, Xiaoyang Wang, Yuan Zhuang, Lin Qiu, and Xun Wang. 2023. Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection. *IEEE Transactions on Knowledge and Data Engineering* 35, 2 (2023), 1296–1310.
- [21] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 254–269.
- [22] S. Meisami and W. E. Bodell. 2023. A Comprehensive Survey of Upgradeable Smart Contract Patterns. arXiv preprint.
- [23] M. Milo, E. H. Nielsen, D. Annenkov, and B. Spitters. 2022. Finding Smart Contract Vulnerabilities with ConCert's Property-Based Testing Framework. arXiv preprint arXiv:2208.00758.
- [24] P. Momeni, Y. Wang, and R. Samavi. 2019. Machine Learning Model for Smart Contracts Security Analysis. In *Proc. International Conference on Privacy, Security and Trust (PST)*. IEEE, Piscataway, NJ, USA, 1–6.
- [25] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. White paper.
- [26] National Institute of Standards and Technology. 2024. Post-Quantum Cryptography Standardization. Accessed: 2024-11-15. <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization>
- [27] Matthew J. Page, Joanne E. McKenzie, Patrick M. Bossuyt, Isabelle Boutron, Tammy C. Hoffmann, Cynthia D. Mulrow, et al. 2021. The PRISMA 2020 Statement: An Updated Guideline for Reporting Systematic Reviews. *BMJ* 372 (2021), n71.
- [28] Parsifal. [n. d.]. About Parsifal: Perform Systematic Literature Reviews. <https://parsifal.ai/about/>. Accessed: 2026-06-12.
- [29] Daniel Perez and Benjamin Livshits. 2021. Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited. In *Proc. USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 1325–1341.
- [30] P. Qian, Z. Liu, Q. He, B. Huang, D. Tian, and X. Wang. 2022. Smart Contract Vulnerability Detection Technique: A Survey. arXiv preprint.
- [31] N. F. Samreen and M. H. Alalfi. 2021. A Survey of Security Vulnerabilities in Ethereum Smart Contracts. *CoRR*, abs/2105.06974.
- [32] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu. 2024. GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. *Proc. IEEE/ACM International Conference on Software Engineering (ICSE)*.
- [33] P. Swathi and D. Bosovic. 2022. A Survey on Quantum-Safe Blockchain System. *Proc. Annual Computer Security Applications Conference (ACSAC)*. ACM.
- [34] Nick Szabo. 1994. Smart Contracts. Unpublished manuscript.
- [35] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.
- [36] C. Wu, L. Chen, K. Wang, W. Han, and H. Chai. 2025. SandWatch: Towards Detecting Sandwich Attacks in Ethereum Using a Dual-Task Graph Neural Network. *ACM Transactions on the Web*.
- [37] H. Wu, Q. Yao, Z. Liu, B. Huang, Y. Zhuang, H. Tang, and E. Liu. 2024. Blockchain for Finance: A Survey. *IET Blockchain*.
- [38] Meihua Xiao, Yangping Xu, Zehuan Li, and Hongbin Wan. 2024. Advanced Security Auditing Methods for Solidity-Based Smart Contracts. *Electronics* 13, 20 (2024), 4093. doi:10.3390/electronics13204093
- [39] S. Yang, J. Niu, and Y. Zhang. 2024. SoK: AI-Powered Security Analysis of Smart Contract. In *Proc. International Conference on Meta Computing (ICMC)*. IEEE, Piscataway, NJ, USA, 91–98.
- [40] L. Zhang, J. Wang, W. Wang, Z. Jin, Y. Su, and H. Chen. 2022. SmartBERT: Vulnerability Detection for Smart Contracts via Pre-Training. arXiv preprint.
- [41] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran. 2019. An Overview on Smart Contracts: Challenges, Advances and Platforms. *Future Generation Computer Systems* 105 (2019), 475–491.
- [42] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais. 2023. SoK: Decentralized Finance (DeFi) Attacks. *Proc. IEEE Symposium on Security and Privacy (S&P)*.
- P2. Full-Stack Hierarchical Fusion of Static Features for Smart Contracts Vulnerability Detection. *IEEE International Conference on Blockchain (Blockchain 2021)*, 2021. <https://doi.org/10.1109/Blockchain53845.2021.00091>
- P3. TokenCheck: Towards Deep Learning Based Security Vulnerability Detection in ERC-20 Tokens. *IEEE Region 10 Symposium (TENSYP 2021)*, 2021. <https://doi.org/10.1109/TENSYP52854.2021.9550913>
- P4. EtherGIS: A Vulnerability Detection Framework for Ethereum Smart Contracts Based on Graph Learning Features. *46th IEEE Annual Computers, Software, and Applications Conference (COMPSAC 2022)*, 2022. <https://doi.org/10.1109/COMPSAC54236.2022.00277>
- P5. Reentrancy Vulnerability Detection of Smart Contract Based on Bidirectional Sequential Neural Network with Hierarchical Attention Mechanism. *International Conference on Blockchain Technology and Information Security (ICBTIS 2022)*, 2022. <https://doi.org/10.1109/ICBTIS55569.2022.00024>
- P6. Smart Contract Vulnerability Detection Based on Abstract Syntax Tree. *8th International Symposium on System Security, Safety, and Reliability (ISSSR 2022)*, 2022. <https://doi.org/10.1109/ISSSR56778.2022.00032>
- P7. Smart Contract Vulnerability Detection Based on Critical Combination Path and Deep Learning. *12th International Conference on Communication and Network Security (ICNS 2022)*, 2022. <https://doi.org/10.1145/3586102.3586135>
- P8. Smart Contract Vulnerability Detection Method Based on Bi-LSTM Neural Network. *IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA 2022)*, 2022. <https://doi.org/10.1109/AEECA55500.2022.9918922>
- P9. A Multi-code Representation Fusion Smart Contract Vulnerability Line Detection Method Based on Graph Neural Network. *11th International Conference on Information Systems and Computing Technology (ISCTech 2023)*, 2023. <https://doi.org/10.1109/ISCTech60480.2023.00013>
- P10. A Multimodal Deep Learning Approach for Efficient Vulnerability Detection in Smart Contracts. *IEEE Global Communications Conference (GLOBECOM 2023)*, 2023. <https://doi.org/10.1109/GLOBECOM54140.2023.10437295>
- P11. A New Smart Contract Anomaly Detection Method by Fusing Opcode and Source Code Features for Blockchain Services. *IEEE Transactions on Network and Service Management*, 20(4), 2023. <https://doi.org/10.1109/TNSM.2023.3278311>
- P12. Automated Smart Contract Vulnerability Detection using Fine-tuned Large Language Models. *6th International Conference on Blockchain Technology and Applications (ICBTA 2023)*, 2023. <https://doi.org/10.1145/3651655.3651658>
- P13. Can Neural Networks Help Smart Contract Testing? An Empirical Study. *14th Asia-Pacific Symposium on Internetware (Internetware 2023)*, 2023. <https://doi.org/10.1145/3609437.3609455>
- P14. Cross-Modality Mutual Learning for Enhancing Smart Contract Vulnerability Detection on Bytecode. *Proceedings of the ACM Web Conference 2023 (WWW 2023)*, 2023. <https://doi.org/10.1145/3543507.3583367>
- P15. Large Language Model-Powered Smart Contract Vulnerability Detection: New Perspectives. *5th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA 2023)*, 2023. <https://doi.org/10.1109/TPS-ISA58951.2023.00044>
- P16. PSCVFinder: A Prompt-Tuning Based Framework for Smart Contract Vulnerability Detection. *IEEE 34th International Symposium on Software Reliability Engineering (ISSRE 2023)*, 2023. <https://doi.org/10.1109/ISSRE59848.2023.00030>
- P17. Semantic-Enriched Code Knowledge Graph to Reveal Unknowns in Smart Contract Code Reuse. *ACM Transactions on Software Engineering and Methodology*, 2023. <https://doi.org/10.1145/3597206>
- P18. Smart Contract Vulnerability Detection Based on TextCNN and Attention Mechanism. *International Conference on Intelligent Management and Software Engineering (IMSE 2023)*, 2023. <https://doi.org/10.1109/IMSE61332.2023.00049>
- P19. A Reliable Framework for Detection of Smart Contract Vulnerabilities for Enhancing Operability in Inter-Organizational Systems. *Journal of Mobile Multimedia*, 20(2), 2024. <https://doi.org/10.13052/jmm1550-4646.2027>
- P20. A Survey of Ethereum Smart Contract Security: Attacks and Detection. *Distributed Ledger Technologies: Research and Practice*, 2024. <https://doi.org/10.1145/3643895>
- P21. Are We There Yet? Unraveling the State-of-the-Art Smart Contract Fuzzers. *46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)*, 2024. <https://doi.org/10.1145/3597503.3639152>
- P22. Checking Smart Contract Vulnerabilities in Blockchain-based Internet of Drones. *5th International Conference on Artificial Intelligence and Computer Engineering (ICAICE 2024)*, 2024. <https://doi.org/10.1145/3716895.3716931>
- P23. VulnHunt-GPT: a Smart Contract Vulnerabilities Detector based on OpenAI ChatGPT. *39th ACM/SIGAPP Symposium on Applied Computing (SAC 2024)*, 2024. <https://doi.org/10.1145/3605098.3636003>
- P24. Combining Deep Learning and Expert Rules for Smart Contract Vulnerability Detection. *IEEE International Conference on Systems, Man, and Cybernetics (SMC 2024)*, 2024. <https://doi.org/10.1109/SMC54092.2024.10831120>
- P25. Empirical Study of Move Smart Contract Security: Introducing MoveScan for Enhanced Analysis. *33rd ACM SIGSOFT International Symposium on Software*

A CATALOGUE OF PRIMARY STUDIES

The 93 primary studies retained by the selection process are catalogued below under the labels **P1–P93** used throughout the categorization tables (Tables 6–8). Studies discussed in depth in the body are additionally cited in the reference list; this catalogue provides the complete and verifiable record of every included study—title, venue, year, and Digital Object Identifier (DOI)—ordered as in the extraction matrix, so that the synthesis and the PRISMA funnel (Figure 2) can be audited end to end.

- P1. Smart Contract Vulnerability Detection Using Code Representation Fusion. *28th Asia-Pacific Software Engineering Conference (APSEC 2021)*, 2021. <https://doi.org/10.1109/APSEC53868.2021.00069>

- Testing and Analysis (ISSTA 2024)*, 2024. <https://doi.org/10.1145/3650212.3680391>
- P26.** Enhancing Smart Contract Security on VNT Blockchain with an Advanced Graph Neural Network Approach. *5th International Artificial Intelligence and Blockchain Conference (AIBC 2024)*, 2024. <https://doi.org/10.1145/3702359.3702361>
- P27.** FunFuzz: A Function-Oriented Fuzzer for Smart Contract Vulnerability Detection with High Effectiveness and Efficiency. *ACM Transactions on Software Engineering and Methodology*, 2024. <https://doi.org/10.1145/3674725>
- P28.** Hierarchical Graph Feature Extraction Based on Multi-Information Contract Graph for Enhanced Smart Contract Vulnerability Detection. *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2024)*, 2024. <https://doi.org/10.1109/TrustCom63139.2024.00221>
- P29.** Identifying Smart Contract Security Issues in Code Snippets from Stack Overflow. *33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2024)*, 2024. <https://doi.org/10.1145/3650212.3680353>
- P30.** Improving Smart Contract Security with Contrastive Learning-based Vulnerability Detection. *46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)*, 2024. <https://doi.org/10.1145/3597503.3639173>
- P31.** OpenTracer: A Dynamic Transaction Trace Analyzer for Smart Contract Invariant Generation and Beyond. *39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024)*, 2024. <https://doi.org/10.1145/3691620.3695361>
- P32.** Optimizing Smart Contract Security: A Cost-Sensitive Graph Neural Network Approach for Vulnerability Detection. *International Conference on Electronic Systems and Intelligent Computing (ICESIC 2024)*, 2024. <https://doi.org/10.1109/ICESIC61777.2024.10846442>
- P33.** SCVHunter: Smart Contract Vulnerability Detection Based on Heterogeneous Graph Attention Network. *46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)*, 2024. <https://doi.org/10.1145/3597503.3639213>
- P34.** SELLM: An Integrated Tool Leveraging Symbolic Execution and LLMs for Smart Contract Vulnerability Detection. *21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP 2024)*, 2024. <https://doi.org/10.1109/ICCWAMTIP64812.2024.10873603>
- P35.** sGuard+: Machine Learning Guided Rule-Based Automated Vulnerability Repair on Smart Contracts. *ACM Transactions on Software Engineering and Methodology*, 2024. <https://doi.org/10.1145/3641846>
- P36.** Smart Contract Code Repair Recommendation based on Reinforcement Learning and Multi-metric Optimization. *ACM Transactions on Software Engineering and Methodology*, 2024. <https://doi.org/10.1145/3637229>
- P37.** Smart Contract Reentrancy Vulnerability Detection Based on CNN and LSTM-Attention. *5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT 2024)*, 2024. <https://doi.org/10.1109/AINIT61980.2024.10581851>
- P38.** Smart Contract Vulnerability Detection Using Deep Learning Algorithms on EVM Bytecode. *13th Mediterranean Conference on Embedded Computing (MECO 2024)*, 2024. <https://doi.org/10.1109/MECO62516.2024.10577852>
- P39.** Smart Contract Vulnerability Detection: The Role of Large Language Model (LLM). *ACM SIGAPP Applied Computing Review*, 24(2), 2024. <https://doi.org/10.1145/3687301.3687305>
- P40.** SoK: AI-Powered Security Analysis of Smart Contract. *International Conference on Meta Computing (ICMC 2024)*, 2024. <https://doi.org/10.1109/ICMC60390.2024.00017>
- P41.** StateGuard: Detecting State Derailment Defects in Decentralized Exchange Smart Contract. *Companion Proceedings of the ACM Web Conference 2024 (WWW 2024)*, 2024. <https://doi.org/10.1145/3589335.3651562>
- P42.** The Sword of Damocles: Upgradeable Smart Contract in Ethereum. *32nd IEEE/ACM International Conference on Program Comprehension (ICPC 2024)*, 2024. <https://doi.org/10.1145/3643916.3644426>
- P43.** TLSCG: Transfer Learning-Based Efficient Anomalous Smart Contract Generation to Empower Unknown Vulnerability Detection. *IEEE International Conference on Web Services (ICWS 2024)*, 2024. <https://doi.org/10.1109/ICWS62655.2024.00068>
- P44.** A Comparative Evaluation of Deep Learning Techniques for Smart Contract Vulnerability Classification. *Distributed Ledger Technologies: Research and Practice*, 4(2), 2025. <https://doi.org/10.1145/3703462>
- P45.** Automated Judging of LLM-based Smart Contract Security Auditors. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2025)*, 2025. <https://doi.org/10.1109/ICBC64466.2025.11114647>
- P46.** Automating Comment Generation for Smart Contract from Bytecode. *ACM Transactions on Software Engineering and Methodology*, 2025. <https://doi.org/10.1145/3699597>
- P47.** Bioluminescent Filament-Inspired AI for Adaptive Smart Contract Intrusion Detection. *9th International Conference on Algorithms, Computing and Systems (ICACS 2025)*, 2025. <https://doi.org/10.1145/3789418.3789441>
- P48.** Blockchain Smart Contract Security: Threats and Mitigation Strategies in a Lifecycle Perspective. *ACM Computing Surveys*, 58(4), 2025. <https://doi.org/10.1145/3769013>
- P49.** Bugdar: AI-Augmented Secure Code Review for GitHub Pull Requests. *IEEE Conference on Artificial Intelligence (CAI 2025)*, 2025. <https://doi.org/10.1109/CAI64502.2025.00113>
- P50.** Characterizing Smart Contract Evolution. *ACM Transactions on Software Engineering and Methodology*, 34(7), 2025. <https://doi.org/10.1145/3719004>
- P51.** CMRL: Cross-Modal Attention and Residual Learning Based Smart Contract Vulnerability Detection. *5th International Conference on Big Data, Artificial Intelligence and Risk Management (ICBAR 2025)*, 2025. <https://doi.org/10.1145/3800227.3800272>
- P52.** CrossMiner: Smart Contract Vulnerability Detection in Interactive Scenarios. *IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2025)*, 2025. <https://doi.org/10.1109/Trustcom66490.2025.00200>
- P53.** DepFuzz: Efficient Smart Contract Fuzzing with Function Dependence Guidance. *Proceedings of the ACM on Programming Languages*, 2025. <https://doi.org/10.1145/3763185>
- P54.** Efficient Adaptation of Large Language Models for Smart Contract Vulnerability Detection. *21st International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2025)*, 2025. <https://doi.org/10.1145/3727582.3728688>
- P55.** Enhancing Smart Contract Security Analysis with Execution Property Graphs. *Proceedings of the ACM on Software Engineering*, 2025. <https://doi.org/10.1145/3728924>
- P56.** Enhancing Smart Contract Security with Explainable AI: A Framework for Reentrancy Vulnerability Detection and Explanation. *Systems and Information Engineering Design Symposium (SIEDS 2025)*, 2025. <https://doi.org/10.1109/SIEDS65500.2025.11021147>
- P57.** Enhancing Smart Contract Security: A Comprehensive Vulnerability Assessment Framework. *2nd International Conference on Computing and Data Science (ICCDs 2025)*, 2025. <https://doi.org/10.1109/ICCDs64403.2025.11209716>
- P58.** Ethereum Smart Contract Security: Deep Learning Approaches for Vulnerability Detection. *3rd International Conference on Sustainable Computing and Data Communication Systems (ICSCDS 2025)*, 2025. <https://doi.org/10.1109/ICSCDS65426.2025.11167498>
- P59.** Evaluating Capabilities and Perspectives of Generative AI Tools in Smart Contract Development. *7th ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI 2025)*, 2025. <https://doi.org/10.1145/3709016.3737802>
- P60.** FIRE: Smart Contract Bytecode Function Identification via Graph-Refined Hybrid Feature Encoding. *16th International Conference on Internetware (Internetware 2025)*, 2025. <https://doi.org/10.1145/3755881.3755883>
- P61.** Fusion Learning for Blockchain Security: Detecting Vulnerabilities in Smart Contracts with GNN and CodeBERT. *9th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS 2025)*, 2025. <https://doi.org/10.1109/CSITSS67709.2025.11295749>
- P62.** Leveraging Large Language Models and Machine Learning for Smart Contract Vulnerability Detection. *IEEE Annual Computing and Communication Workshop and Conference (CCWC 2025)*, 2025. <https://doi.org/10.1109/CCWC62904.2025.10903869>
- P63.** Leveraging Machine Learning Models to Improve Smart Contract Security: A Survey of Vulnerabilities and Detection Methods. *ACM Computing Surveys*, 58(6), 2025. <https://doi.org/10.1145/3772367>
- P64.** LLM Assisted Dual-View Awareness Framework for Smart Contract Vulnerability Detection. *IEEE 36th International Symposium on Software Reliability Engineering (ISSRE 2025)*, 2025. <https://doi.org/10.1109/ISSRE66568.2025.00022>
- P65.** MCLF: Smart Contract Vulnerability Detection Based on Multimodal and Contrastive Learning. *6th International Conference on Artificial Intelligence and Computer Engineering (ICAICE 2025)*, 2025. <https://doi.org/10.1109/ICAICE68195.2025.11382305>
- P66.** Optimized DeBERTa for Efficient Smart Contract Vulnerability Detection. *30th Asia-Pacific Conference on Communications (APCC 2025)*, 2025. <https://doi.org/10.23919/APCC64555.2025.11279833>
- P67.** Performance of Open-Source Large Language Models in Smart Contract Vulnerability Detection. *IEEE International Conference on Blockchain Technology and Information Security (ICBCTIS 2025)*, 2025. <https://doi.org/10.1109/ICBCTIS66509.2025.11387384>
- P68.** Research on Intelligent Vulnerability Detection Model Based on Multi-Modality. *6th International Conference on Computer Engineering and Application (ICCEA 2025)*, 2025. <https://doi.org/10.1109/ICCEA65460.2025.11102232>
- P69.** Research on Multi-feature Fusion Graph Neural Network-based Vulnerability Detection Method for Smart Contracts. *IEEE International Conference on Computing, Communication, Perception and Quantum Technology (CCPQT 2025)*, 2025. <https://doi.org/10.1109/CCPQT66408.2025.11382789>

- P70.** SandWatch: Towards Detecting Sandwich Attacks in Ethereum Using a Dual-Task Graph Neural Network. *ACM Transactions on the Web*, 2025. <https://doi.org/10.1145/3787205>
- P71.** Semantics-Compressed and Attention-Guided Framework for Smart Contract Vulnerability Detection. *5th International Conference on Machine Learning and Intelligent Systems Engineering (MLISE 2025)*, 2025. <https://doi.org/10.1109/MLISE66443.2025.11100212>
- P72.** Smart Contract Fuzzing Towards Profitable Vulnerabilities. *Proceedings of the ACM on Software Engineering*, FSE, 2025. <https://doi.org/10.1145/3715720>
- P73.** Smart Contract Reentry Vulnerability Detection Based on Federated Learning. *International Symposium on Computer Applications and Information Technology (ISCAIT 2025)*, 2025. <https://doi.org/10.1109/ISCAIT64916.2025.11010346>
- P74.** Smart Contract Security: A Comprehensive Evaluation of Deep Learning Based Mechanisms for Smart Contract Vulnerability Detection. *IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS 2025)*, 2025. <https://doi.org/10.1109/ICBDS67396.2025.11377305>
- P75.** Smart-LLaMA-DPO: Reinforced Large Language Model for Explainable Smart Contract Vulnerability Detection. *Proceedings of the ACM on Software Engineering*, ISSTA, 2025. <https://doi.org/10.1145/3728878>
- P76.** SmartAuditFlow: A Dynamic Plan-Execute Framework for Advanced Smart Contract Security Analysis. *ACM Transactions on Software Engineering and Methodology*, 2025. <https://doi.org/10.1145/3785364>
- P77.** SNFM: Scalable Node Feature Modeling in Smart Contracts for Vulnerability Detection through Graph-based Approaches. *IEEE Global Blockchain Conference (GBC 2025)*, 2025. <https://doi.org/10.1109/GBC60041.2025.11134470>
- P78.** SolGen: Secure Smart Contract Code Generation Using Large Language Models Via Masked Prompting. *18th Innovations in Software Engineering Conference (ISEC 2025)*, 2025. <https://doi.org/10.1145/3717383.3717394>
- P79.** SuiGPT MAD: Move AI Decompiler to Improve Transparency and Auditability on Non-Open-Source Blockchain Smart Contract. *Proceedings of the ACM Web Conference 2025 (WWW 2025)*, 2025. <https://doi.org/10.1145/3696410.3714790>
- P80.** Vulnerability Smart Contract Detection Model Based on CUR Matrix Decomposition. *IEEE International Conference on Computation, Big-Data and Engineering (ICCBDE 2025)*, 2025. <https://doi.org/10.1109/ICCBDE65177.2025.11255968>
- P81.** When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We? *ACM Transactions on Software Engineering and Methodology*, 2025. <https://doi.org/10.1145/3702973>
- P82.** A Review of Learning-based Smart Contract Vulnerability Detection: A Perspective on Code Representation. *ACM Transactions on Software Engineering and Methodology*, 35(6), 2026. <https://doi.org/10.1145/3750042>
- P83.** AI-Empowered Smart Contract Vulnerability Detection for Decentralized Blockchain Systems. *IEEE Transactions on Network Science and Engineering*, 2026. <https://doi.org/10.1109/TNSE.2026.3658785>
- P84.** CCIHunter: Enhancing Smart Contract Code-Comment Inconsistencies Detection via Two-Stage Pre-Training. *ACM Transactions on Software Engineering and Methodology*, 2026. <https://doi.org/10.1145/3764867>
- P85.** Digital Human-Assisted Smart Contract Vulnerability Detection Under Limited Sample Constraints. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2026)*, 2026. <https://doi.org/10.1109/ICASSP55912.2026.11461273>
- P86.** Empirical Analysis of Smart Contract Factories on EVM-compatible Chains. *ACM Transactions on Software Engineering and Methodology*, 2026. <https://doi.org/10.1145/3787216>
- P87.** Enhancing Security and Acuity of Smart Contract Vulnerability Detection Based on Federated Learning and BiLSTM-Attention. *ACM Transactions on Software Engineering and Methodology*, 35(4), 2026. <https://doi.org/10.1145/3746061>
- P88.** From Cryptic to Clear-Training on LLM Explanations to Detect Smart Contract Vulnerabilities. *ACM Transactions on Software Engineering and Methodology*, 35(6), 2026. <https://doi.org/10.1145/3765753>
- P89.** MANDO-LLM: Heterogeneous Graph Transformers with Large Language Models for Smart Contract Vulnerability Detection. *ACM Transactions on Software Engineering and Methodology*, 35(6), 2026. <https://doi.org/10.1145/3765751>
- P90.** Multi-Objective-Guided Static Analysis for Smart Contract Vulnerability Detection. *14th International Conference on Software and Information Engineering (ICSIE 2026)*, 2026. <https://doi.org/10.1145/3789595.3789604>
- P91.** No More Hidden Pitfalls? Exposing Smart Contract Bad Practices with LLM-Powered Hybrid Analysis. *ACM Transactions on Software Engineering and Methodology*, 2026. <https://doi.org/10.1145/3795692>
- P92.** ParaVul: A Parallel Large Language Model and Retrieval-Augmented Framework for Smart Contract Vulnerability Detection. *IEEE Transactions on Information Forensics and Security*, vol. 21, 2026. <https://doi.org/10.1109/TIFS.2026.3694661>
- P93.** Smart Contracts and Security: A Comprehensive Survey of Vulnerabilities. *IEEE 23rd Mediterranean Electrotechnical Conference (MELECON 2026)*, 2026. <https://doi.org/10.1109/MELECON64486.2026.11418841>