# SoK: AI-Powered Security Analysis of Smart Contract

Shuo Yang[1,2], Jianyu Niu[1], Yinqian Zhang[1]
[1]Southern University of Science and Technology [2]Chinese University of Hong Kong
1155198056@link.cuhk.edu.hk, niujy@sustech.edu.cn, yinqianz@acm.org

***Abstract*—With the soaring popularity of decentralized applications (DApps), smart contract security has become increasingly important. Recently, numerous studies have leveraged artificial intelligence (AI) techniques to enhance efficiency and functional diversity of smart contract security analysis. However, a comprehensive survey of these studies to guide future development is still missing. To fill this gap, we present an innovative and systematic review. First, we establish filtering criteria and define Literature Attributes (LA) to identify 27 representative papers in this field. We then trace their evolution from 2018 and summarize four key research problems. Next, we compare three AI-powered smart contract security analysis tools—DLVA, xFuzz, and GURU—against traditional methods, demonstrating that AI tools still have room for improvement in precision. Finally, we discuss opportunities for improving AI-powered smart contract security analysis.**

***Keywords*—Blockchain, smart contract, artificial intelligence, program analysis**

## I. Introduction

Blockchain technology has become a transformative force in digital interaction. Smart contracts—autonomous and self-executing programs deployed on blockchains—are crucial to Decentralized Finance (DeFi) [1], supply chain [2], Web3 DApps [3], and Metacomputing [4]. However, the rapid evolution of the smart contract has raised extensive security concerns. Once a smart contract is deployed on blockchains, its code becomes unalterable, thus precluding any post-deployment revisions or withdrawals. Consequently, any inherent vulnerabilities within the smart contract can lead to irreversible financial damages. For example, a notable security breach in the Ethereum smart contract resulted in the theft of approximately 3.6 million Ether [5]. Besides, Nicola Atzei's preliminary analysis of 19,366 Ethereum smart contracts shows that a staggering 8,833 have vulnerabilities [6].

Due to its importance, security issues related to smart contracts are gaining increasing attention. For example, extensive studies using traditional analysis methods such as customized pattern rules [7], symbolic execution [8], taint analysis [9], and fuzz testing [10], [11] emerges. Beyond these, studies such as TXSPECTOR [12] and DEFIER [13] also analyze on-chain transaction information to detect malicious behaviors. More importantly, AI technology has brought a new perspective to smart contract security analysis. Prominent examples include SAFERSC [14], SMARTINV [15], and BlockGPT [16]. These studies utilize Machine Learning (ML), Deep Learning (DL), and other AI technology to automatically detect vulnerabilities and identify malicious attacks. What is more, the emergence of Large Language Models (LLMs) can provide new possibilities for vulnerability identification and autonomous code audits for their powerful text generation and reasoning capabilities. Compared with traditional methods, AI technology brings both new opportunities and challenges to this research direction.

Despite the fast development, AI-powered security analysis of smart contracts is still in its infancy. There is no systematic study of research problems, technical methods, and evaluation systems, that summarize the current state, identify shortcomings, and highlight challenges for future research. Besides, analyzing applicable scenarios and finding limitations of AI technology can help promote further development of AI, Web3, and Meta computing. All of these motivate us to fill this review gap by providing a useful reference for research on AI-powered smart contract security analysis technology in the era of Web3.

In this paper, we conduct a systematic analysis of AI-powered smart contract security studies from 2018 to 2024. We select 27 representative studies and perform a thorough manual review. After that, we identify four primary research directions from the program analysis perspective: 1) smart contract vulnerability detection, 2) anomalous smart contract detection, 3) smart contract security analysis enhancement, and 4) smart contract reverse engineering. (See details of each category in Sec. IV-B.) We also examine data collection and feature engineering of different studies. Despite variations in research objectives, we observe a high level of consistency in data collection and feature engineering. However, this homogeneity may limit the effectiveness of AI models in specific tasks. Furthermore, we identify four directions that have garnered widespread attention, reflecting the focus and development trends in AI-powered smart contract security analysis.

We evaluate and compare the performance of three tools—xFuzz [17], DLVA [18], and GURU [19]—-in detecting reentrancy vulnerabilities. We observe that existing AI-powered analysis tools have better efficiency in vulnerability detection due to the end-to-end classification, however, lower precision compared to traditional analysis tools. This highlights the potential of integrating traditional program analysis with AI

Jianyu Niu and Yinqian Zhang are affiliated with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering of Southern University of Science and Technology (SUSTech).

TABLE I
COMMON SMART CONTRACT VULNERABILITIES

| Layer | Vulnerability |
|---|---|
| Code | Reentrancy [25]–[27]<br>Access Control [28], [29]<br>Arithmetic Overflow/Underflow [30]<br>Unchecked External Calls [31]<br>Denial of Service (DoS) [32] |
| Execution environment | Short Address Attack<br>Call Stack Overflow |
| Blockchain system | Timestamp Dependency<br>Transaction Ordering |

technology, leveraging the interpretability and accuracy of traditional methods alongside the efficiency of AI technology.

**Contributions.** The main contributions are as follows:

- We filter 27 high-quality studies on AI-powered smart contract security analysis, which are collected from 15 top journals and conferences.
- We analyze and summarize the specific research content of these studies, identifying four potential sub-research directions: 1) Smart Ponzi Scheme, 2) Vulnerability Detection, 3) Code Comment Generation, and 4) Invariant Inference.
- We evaluate three open-source tools. Furthermore, we also assess four existing tools and identify their limitations from the user's perspective.

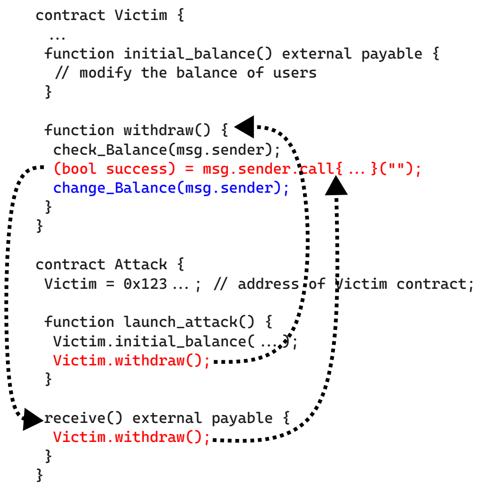
## II. BACKGROUND

### A. Smart Contract

Smart contracts are programs running on blockchains. Once deployed on blockchains, a smart contract refers to a collection of code and data with a specific blockchain address [20]. Otherwise, it represents code files written in high-level programming languages (e.g., Solidity [21] and Rust [22] ). Smart contracts have the following key features: 1) autonomy [23], which means smart contracts operate independently without intermediaries; 2) interoperability [24], by which they can interact with each other, enabling the development of complex, integrated operations; and 3) customizability, which means smart contracts are programmed to suit various applications, offering versatile solutions across various sectors.

**Security Vulnerabilities.** Adversaries can utilize smart contract code to launch attacks or steal assets, resulting in security vulnerabilities. These vulnerabilities are mainly caused by three aspects: code layer, execution environment layer, and blockchain system layer [33]. Table I shows some common smart contract vulnerabilities. Due to space constraints, we only introduce reentrancy vulnerability in detail.

The reentrancy vulnerability occurs when a function makes external calls to untrusted contracts. Malicious adversaries can exploit these calls to re-enter the original function, alter the contract's state, or withdraw funds before the initial execution is completed. All of these lead to unauthorized state changes.



```
contract Victim {
 ...
 function initial_balance() external payable {
  // modify the balance of users
 }

 function withdraw() {
  check_Balance(msg.sender);
  (bool success) = msg.sender.call{ ... }("");
  change_Balance(msg.sender);
 }
}

contract Attack {
 Victim = 0x123 ... ; // address of Victim contract;

 function launch_attack() {
  Victim.initial_balance( ... );
  Victim.withdraw();
 }

 receive() external payable {
  Victim.withdraw();
 }
}
```

Fig. 1. An Example of Reentrancy Vulnerability

The most famous reentrancy attack on DAO has caused a loss of $60 million [25].

Fig. 1 illustrates an example, in which the attacker deploys a smart contract with malicious code (denoted by $C_{adv}$). The contract $C_{adv}$ first deposits some funds into the victim contract (denoted by $C_{vic}$), and then requests to withdraw their funds from $C_{vic}$. After $C_{vic}$ sends funds to $C_{adv}$, $C_{adv}$ sends a new withdrawal request before it updates the attacker's balance according to the previous withdrawal request. Since $C_{vic}$ has not yet updated the balance, $C_{adv}$ processes another withdrawal. Furthermore, by carrying the above loop, the attacker can repeatedly transfer funds from $C_{vic}$ to $C_{adv}$ until $C_{vic}$'s funds are drained.

### B. Smart Contract Vulnerabilities

**Access Control.** Access control vulnerability arises when sensitive contract variables or functions are incorrectly exposed as public. This oversight allows any user to modify the contract's state or invoke functions that should be restricted, compromising the contract's integrity and security.

**Arithmetic Overflow/Underflow.** Arithmetic overflow and underflow happen when numerical operations exceed the limits of the variable type assigned to store the result. This can cause integers to wrap around and produce incorrect values, leading to logical errors and potential exploitation. However, the programming language Solidity v0.8.x prevents this problem at the code level through an auto-recovery mechanism.

**Unchecked External Calls.** Unchecked external calls vulnerability refers to the failure of a contract to validate the call address of external function calls properly. As a result, a contract could continue execution even when calls to untrusted contracts, potentially leading to unauthorized changes in the contract state.

**DoS.** DoS in the smart contract can be caused by logical errors or unhandled exceptions that disrupt the normal execution

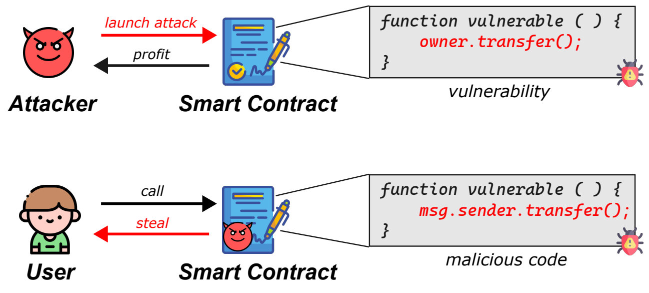


Fig. 2. Two Different Smart Contract Security Analysis Scenarios

flow. This can freeze the contract's functions, making it unable to process transactions or interact with users as intended.

## C. Smart Contract Security Analysis

Smart contract security analysis targets vulnerabilities in smart contracts before deployment and attacks on deployed smart contracts. We categorize smart contract security analysis into two scenarios: analyzing vulnerabilities of smart contracts and malicious smart contracts, as shown in Fig. 2.

**Analyzing Vulnerabilities in Smart Contracts.** Attackers exploit the *vulnerable* functions of smart contracts to illegally transfer funds, posing significant financial risks once these contracts are deployed. Thus, in this paper, we focus on analyzing vulnerabilities before a contract is deployed.

**Analyzing Malicious Smart Contracts.** The analysis of malicious smart contracts focuses on identifying harmful operations. They may disguise themselves as normal contracts. Once a user interacts with such a contract, malicious code can trigger asset theft or other malicious activities. As an example, Fig. 2 show that the smart contract contains malicious code *msg.sender.transfer()*. When a user calls this function, their funds can be transferred to the attacker. This kind of analysis is mainly conducted on deployed smart contracts.

## III. Review Methodology

### A. Research Questions

We define four research questions to systematically investigate prior studies on AI-powered smart contract security analysis. Specifically, there are one general question (GQ) and three research questions (RQ):

- **GQ1**: What are some representative studies of AI-powered smart contract security analysis, and what is their content?
- **RQ1**: What research directions are included in the field of AI-powered smart contract security analysis?
- **RQ2**: How do these studies collect and process the data used for AI model training?
- **RQ3**: Do these studies share a common problem?

To clarify the main logical thread of smart contract security analysis research, we correlate the three analytical stages: research contents (GQ1), research directions (RQ1), and the specific common research problems (RQ3). Here, GQ1 outlines the main areas of focus in current studies; RQ1 reveals the specific implementation methods of these activities from a more macroscopic perspective; RQ3 refines the hot issues within the research content and direction.

### B. Research Subjects

We first collect 204 relevant papers from 2018 to 2024 by searching targeted keywords in top peer-reviewed scientific databases (e.g., IEEE, ACM, and Elsevier). Next, we manually review these papers, filtering them according to pre-established criteria to exclude irrelevant and review-type papers. Thereafter, we identify 15 target journals and conferences as benchmarks and select high-quality research papers as well as their relevant work. What is more, to ensure objectivity and fairness in the paper selection, this review specified six criteria: 1) The paper has a clear purpose of using AI; 2) AI technology is used; 3) The paper targets smart contract security analysis; 4) Effectiveness of the technical methods is validated; 5) The paper provides clear research conclusions. and 6) Research results are summarized and future research directions are proposed. With the criteria, we made an information extraction template [34] to filter papers.

## IV. Review Result

### A. GQ: Representative Studies

Table II provides a comprehensive summary of studies on AI-powered smart contract security analysis. It categorizes the studies by four key attributes: publication year, proposed tool, research content, and the analysis level (high-level source code or compiled bytecode). The publication year highlights recent trends in the field. The research content is summarized in four categories: 1) Vulnerability Detection, 2) Malicious Detection, 3) Code Synthesis, and 4) Comment Synthesis. The analysis level indicates the specific focus of each research study.

It is noteworthy that the launch of the LLM in the series of GPT, a revolutionary change in the field of AI, brought new perspectives and methods to the AI-powered smart contract security analysis. Thus, this review divides the associated studies into two stages: *Before GPT (from 2018 to 2022)* and *After GPT (from 2022 to present)*. This division not only helps to highlight the impact of the LLM on the field of AI-powered smart contract security analysis but also allows us to more clearly observe the influence of the development of AI technology on the research direction and methodology of smart contracts security analysis.

*1) Before GPT:* At this stage, AI-powered smart contract security analysis is in its infancy, characterized by relatively simplistic technologies and code features, and similar research content. The primary content of these studies is to use code features of smart contracts to predict vulnerabilities.

DPS [35] demonstrates the effectiveness of using the N-gram language model, a traditional statistical model, for smart contract representation. It combines bytecode and transaction information as code features to detect Ponzi schemes in smart contracts (smart Ponzi scheme). Similarly, S-gram [36] applies the N-gram language model to predict smart contract vulnerabilities by modeling irregular token sequences in transactions.

TABLE II
REPRESENTATIVE RESEARCH ON SMART CONTRACT SECURITY COMBINED WITH AI TECHNOLOGY

| Year | Tool | Research Content | Analysis Level |
|---|---|---|---|
| 2018 | DPS [35] | Malicious Perception | Source Code |
| 2018 | S-gram [36] | Vulnerability Detection | Source Code |
| 2018 | SAFERSC [14] | Vulnerability Detection | Source Code |
| 2020 | Contract-Ward [37] | Vulnerability Detection | Source Code, Bytecode |
| 2020 | Smartembed [38] | Vulnerability Detection | Source Code |
| 2021 | GSCVD [39] | Vulnerability Detection | Source Code |
| 2021 | GSCV [40] | Vulnerability Detection | Source Code |
| 2021 | SmarTest [41] | Vulnerability Detection | Source Code |
| 2021 | HVSC [42] | Vulnerability Detection | Bytecode |
| 2021 | EOSAFE [43] | Vulnerability Detection | Bytecode |
| 2021 | SmartDoc [44] | Comment Synthesis | - |
| 2021 | Al-SPSD [45] | Malicious Perception | Bytecode |
| 2022 | Cider [46] | Vulnerability Detection | Source Code |
| 2022 | Rlf [47] | Vulnerability Detection | - |
| 2022 | xFuzz [17] | Vulnerability Detection | Source Code |
| 2022 | GURU [19] | Vulnerability Detection | Source Code |
| 2023 | DLVA [18] | Vulnerability Detection | Bytecode |
| 2023 | Tx2txt [48] | Comment Synthesis | - |
| 2023 | MFSCV [49] | Vulnerability Detection | Source Code, Bytecode |
| 2023 | MulCas [50] | Vulnerability Detection | Source Code, Bytecode |
| 2023 | Vcd [51] | Code Synthesis | Source Code |
| 2023 | GPTLENS [52] | Vulnerability Detection | Bytecode |
| 2023 | GPTSCV [53] | Vulnerability Detection | Bytecode |
| 2023 | BlockGPT [16] | Malicious Perception | Transactions |
| 2023 | GPTscan [54] | Vulnerability Detection | Bytecode |
| 2024 | SMARTINV [15] | Vulnerability Detection | Source Code |
| 2024 | PonziGuard [55] | Malicious Perception | Source Code, Bytecode |

To further extend the semantic context scale, SAFERSC [14] explores the potential of encoding more extensive information from code and transaction information using DL rather than the N-gram model. These approaches exhibit limitations such as strong specificity, limited analysis generalization capabilities, and reliance on expert knowledge.

To further enhance the analysis generalization, some studies approach it similarly to more solvable tasks. Contract-Ward [37] treats smart contract security analysis as a classification task, aiming to detect a wider range of common vulnerabilities. Smartembed [38] addresses it as a similarity detection task by converting smart contract source code into numerical vectors using word embedding, detecting vulnerabilities by determining the similarity between smart contracts thereby reducing the reliance on expert knowledge.

Since smart contracts are structured in a form that can be represented graphically, several studies, including GSCV [40], GSCVD [39], and HVSC [42], employ Graph Neural Networks (GNNs) to represent and model smart contract inputs in a structured format. These approaches integrate various code features into a unified representation, thereby enhancing the analysis generalization.

Numerous studies provide further insights into the AI-powered security analysis of smart contracts, focusing on a variety of research questions and objectives. Al-SPSD [45] addresses the data distribution imbalance issue caused by the homogenization of smart contracts. EOSAFE [43] concentrates on identifying vulnerabilities in EOSIO smart contracts. SmarTest [41] focuses on analyzing vulnerable transactions. SmartDoc [44] introduces the task of automatically generating code comments for smart contracts to enhance users' understanding of their semantics.

*2) After GPT:* From 2022 to 2024, studies on AI-powered smart contract security analysis evolve from relying on a single technique to integrating multiple techniques, and also shift from general-purpose research problems (e.g., the pursuit of detecting more types of vulnerabilities) to a deeper focus on specific research problems (e.g., invariant inference and code comment generation). This research evolution is primarily evident in the following aspects:

The powerful understanding and reasoning capabilities of LLMs provide new opportunities for smart contract security analysis. GPTSCV [53] and GPTscan [54] conduct preliminary validations of using LLMs in smart contract vulnerability detection, laying the groundwork for further studies. SMARTINV [15] utilizes LLMs to infer invariants for symbolic execution. Vcd [51], and GPTLENS [52] directly detect vulnerable smart contract code fragments based on LLMs.

Second, the capability of GNNs to capture the semantics of smart contracts is continuously applied and developed. Studies like GURU utilize GNNs to process heterogeneous graph representations of smart contracts, including Control-flow Graphs (CFG) and call graphs. Similarly, PonziGuard [55] employs GNNs to model runtime behavior graphs. This methodology enables precise modeling of contract semantics and accurate localization of vulnerabilities.

Some studies also explore integrating AI technology with traditional program analysis. For reinforcement learning (RL), Cider [46] applies it to invariant inference; Rlf processes of fuzzing smart contracts as a Markov decision process to apply RL into fuzz testing. For ML, xFuzz uses a model trained by word vectors and instructions to filter likely benign program paths to guide fuzz testing.

In addition to combining AI technology with traditional program analysis, there is a trend toward integrating multiple AI technologies. DLVA and MFSCV [49] both introduce a multimodal AI framework that effectively utilizes information from different layers, such as source code, compilation data, and bytecode, to extract joint multimodal feature representations. This approach significantly enhances the precision of the analysis.

The research on smart contract security analysis from 2022 to 2024 shows a development trend of multi-technology integration and multi-perspective analysis. Introducing advanced AI technologies such as LLM and GNN provides new ideas and tools for smart contract security analysis.

**Summary.** The research from 2018 to 2022 mainly uses preliminary AI technologies (e.g., N-gram language model) and code features (e.g., source code patterns and bytecode fragments) for smart contract security analysis. Since 2022, LLM has marked a qualitative leap in AI-powered smart contract security analysis. In addition, there is a growing trend towards composite research incorporating various AI technology and traditional program analysis.

### *B. RQ1: Research Directions in Smart Contract Security Analysis*

We analyze the 27 representative studies mentioned earlier and categorize them into four research directions based on traditional program analysis.

**Smart Contract Vulnerability Detection.** A significant body of studies is dedicated to the automatic detection of various smart contracts vulnerabilities. Notable examples of this research in this direction include DPS, S-gram, Contract-Ward, Smartembed, GSCV, GSCVD, HVSC, among others. These studies focus on identifying issues such as reentrancy attacks, transaction order dependencies, and other security threats.

**Anomalous Smart Contract Detection.** Studies such as PonziGuard and MulCas aim to precisely detect smart Ponzi schemes using pre-trained models and GNNs.

**Smart Contract Security Analysis Enhancement.** This type of study focuses on using AI technology to handle certain time-consuming or resource-intensive tasks in the process of smart contract security analysis, such as invariant inference during formal verification and path filtering during fuzz testing, thereby improving the efficiency of smart contract security analysis. Studies like Cider, Rlf, and xFuzz are dedicated to advancing development in this direction.

**Smart Contract Reverse Engineering.** When only the bytecode of a smart contract is available, recovering higher-level semantic information from the bytecode provides richer insights for smart contract security analysis. For instance, SmartDoc explores methods for automatically generating smart contract code comments. Similarly, Tx2txt investigates this direction, aiming to recover the high-level semantics of smart contract bytecode fragment.

### *C. RQ2: Data Collection and Processing*

Table III presents a comparative analysis of data collection and processing methods used in nine selected pieces of research. The analysis focuses on tagged objects, data sources, tagging tools (tag makers), bias removal techniques, code features, and baseline models.

**Tagged objects.** The research shares similar characteristics in that they all collect real-world smart contracts from Etherscan. Some studies preprocess the collected smart contracts to extract their abstract syntax trees (AST), functions, and opcodes. However, the studies differ in data labeling and bias removal processes. Only six studies describe their labeling methods, with five using automated static analysis tools to classify vulnerability labels. Additionally, only four studies explicitly mention their bias removal strategies.

**Feature engineering.** Eight studies use opcodes as the primary objects, while one extracts features from ASTs. The choice of baseline models varies significantly across the studies, with SPC and Contract-Ward using eXtreme Gradient Boosting (XGBoost) and others employing a diverse range of models.

In summary, although data collection methods are consistent, there is significant variation in labeling, bias removal, feature engineering, and baseline model selection. These findings highlight the need for more standardized and transparent data processing practices to enhance the comparability and reproducibility of results.

### *D. RQ3: Common Research Problems*

*1) Smart Ponzi Scheme:* A smart Ponzi scheme is a type of malicious smart contract in a financial sense. Corresponding to IV-B, analyzing a smart Ponzi scheme belongs to a kind of anomalous smart contract detection. Four studies [35], [45], [50], [55] focus on this research problem. In the initial research phase, DPS uses specific vulnerable bytecode patterns as the code feature to train the classification model to predict the smart Ponzi scheme. Recent research, such as PonziGuard, further incorporates smart contract runtime information (a.k.a., Contract Runtime Behavior Graph) into code features and formulates the problem as a graph classification task, improving the detection performance with richer code features and a more generalized modeling method.

*2) Vulnerability Detection:* This is one of the mainstream research problems in the field of smart contracts. Initially, the studies simply model the task of vulnerability detection as a classification problem. As the ecosystem of smart contracts continues to evolve, related studies have gradually improved in terms of technical complexity (e.g., using multimodal and LLMs), the depth of the issues (expanding from mere vulnerability detection to a variety of related problems), and the efficiency and accuracy of vulnerability detection. Additionally, recent studies, such as DLVA, have also focused on improvements in datasets and experimental methods, emphasizing the need for comprehensive evaluation of proposed methods on real-world datasets.

*3) Code Comment Generation:* Tx2txt generates intermediate representations based on bytecode, providing a natural language description of the smart contract's operating logic, which aids in security analysis. SmartDoc focuses on directly generating user-readable function comments. Both employ Seq2Seq models from Natural Language Processing (NLP) and improve the design of encoding-decoding networks and training paradigms.

Despite some encouraging progress, there is still a lack of follow-up research to validate the practical benefits of such code comments for smart contract security analysis.

*4) Invariant Inference:* In formal verification, inferring smart contract invariants is crucial yet challenging. Cider trains the agent to infer invariants based on RL, using the smart contract verifier as the simulation environment of the agent training process. SMARTINV, on the other hand, makes full use of the inference capability of LLMs and realizes invariant inference based on LLMs by encoding the expert knowledge of the inference invariant into the prompt.

### *E. Summary of Answers for Questions*

This review investigates research on AI-powered smart contract security analysis and draws the following conclusions.

TABLE III
DATA PROCESSING METHODS AND BASELINE USED IN TRAINING MODELS FOR REPRESENTATIVE TOOLS. SC FOR SOURCE CODE, TX FOR TRANSACTION, MI FOR MANUAL INSPECTION, TI FOR TOOL INSPECTION, CD FOR CODE DEDUPLICATION

| **Tool** | **Data Collection** | | | | **Data Feature** | **Baseline Algorithm/Model** |
|---|---|---|---|---|---|---|
| | **Tagged Objects** | **Data Source** | **Tag Maker** | **Bias Removal** | | |
| SPC | SC, TX | Etherscan | - | - | Opcode,Account | XGBoost |
| AI-SPSD | SC | Etherscan, Datasets, DApp | *MI, TI* | *CD* | Opcode | Decision Tree |
| PonziGuard | SC | Etherscan, Datasets | *MI* | - | runtime information | GNN |
| SAFERSC | Opcode | Etherscan | *TI* | *MI, CD* | Opcode | Long Short-Term Memory (LSTM) |
| Contract-Ward | SC | Etherscan | *TI* | - | Opcode | XGBoost |
| SMARTMBED | SC, AST | Etherscan | - | - | AST | FastText |
| DLVA | SC | Etherscan | - | - | CFG | USE |
| xFuzz | Function | Etherscan | *TI* | *MI* | Opcode | EasyEnsembleClassifier (EEC) |
| SMARTINV | SC | Etherscan, DApp | *MI, TI* | *MI, CD* | Opcode | LLaMA |

**GQ1**: Representative studies on AI-powered smart contract security analysis are temporally distributed over seven years from 2018 to the present. We find 27 high-quality studies, as shown in Table II.

**RQ1**: There are four types of research directions in AI-powered smart contract security analysis: 1) smart contract vulnerability detection, 2) anomalous smart contract detection, 3) smart contract security analysis enhancement, and 4) smart contract reverse engineering.

**RQ2**: Although there are significant differences in the data processing methods used by different researchers during the data collection period. For instance, SMARTINV employs a dual method of manual inspection and tool inspection to annotate data. Most studies collect smart contract source code and bytecode through Etherscan and primarily extract code features from the opcode.

**RQ3**: The common research problems of AI-powered smart contract security analysis include: 1) smart Ponzi scheme, 2) vulnerability detection, 3) code comment generation, and 4) invariant inference.

## V. EVALUATION

In this section, we select representative studies to illustrate the impact of AI technology on smart contract security analysis tools. We compare their vulnerability detection capabilities with each other and with traditional smart contract security analysis tools, using evaluation data for analysis. All experiments are conducted on an Ubuntu 22.04 LTS computer with an Intel i5 9300H 2.4GHz processor, 32GB of RAM, and a 1.5TB HDD.

### A. Evaluation Object Selection

To assess the effectiveness of AI technology in smart contract security analysis, we focus on three tools: xFuzz, DLVA, and GURU. xFuzz integrates ML with fuzz testing to optimize path filtering. DLVA showcases the power of combining various AI models for in-depth analysis. At the same time, GURU leverages GNN for vulnerability detection. We use Oyente, Slither, sFuzz and Confuzzius integrated by SmartBugs [56] as baselines to compare with these three tools.

### B. Experimental Problems

We design experiments to assess the tools' precision, recall, and efficiency in vulnerability detection. These experiments aim to answer the following questions:

- **Recall and Precision.** How accurate and comprehensive are the AI-powered smart contract security analysis tools in identifying vulnerabilities compared to traditional methods?
- **Efficiency.** Do AI technology approaches enhance the speed of smart contract security analysis?
- **Utility.** What is the utility of AI-powered smart contract security analysis tools?

By addressing these questions, we can gain valuable insights into the effectiveness and practicality of integrating AI technology into smart contract security analysis.

### C. Metrics

To quantitatively assess the performance of the selected tools, we employ several metrics:

*1) Recall and Precision:* We use the confusion matrix parameters True Positive (*TP*), False Positive (*FP*), True Negative (*TN*), and False Negative (*FN*) to calculate *precision* ($TP/(TP+FP)$) and *recall* ($TP/(TP+FN)$). *Precision* measures the proportion of *TP* vulnerabilities among all detected vulnerabilities, while *Recall* represents the percentage of actual vulnerabilities successfully identified by the tool. Additionally, we introduce the Analysis Failure (*AF*) metric to gauge the success rate of smart contract security analysis by each tool. *AF* accounts for instances where the tool fails to complete the analysis process.

*2) Efficiency:* To evaluate the efficiency of these three tools, we introduce True Positive Time (*TPT*), which represents the average amount of time taken for a tool to identify a TP case. Thus, we have $TPT = T/TP$, where $T$ is the total analysis time, and *TP* is the number of true positives discovered. We also calculate the mean analysis time (*MAT*) for each contract as $MAT = T/n$, where $T$ is the total analysis time, and $n$ is the number of analyzed contracts.

### D. Vulnerability Selection

We focus on the tools' performance in detecting reentrancy vulnerabilities for two reasons. First, The types of supported

TABLE IV
PRECISION AND RECALL OF AI-POWERED SMART CONTRACT SECURITY ANALYSIS TOOLS COMPARING WITH SOTA IN TRADITIONAL TOOLS

| Tool | Dataset 1 (size = 41) | | | | | | Dataset 2 (size = 653) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AF | TP | TN | FP | FN | Recall | AF | TP | TN | FP | FN | Recall | Precison |
| GURU | 11 | 30 | 0 | 0 | 0 | 100.00% | - | - | - | - | - | - | - |
| DLVA | 0 | 13 | 0 | 0 | 28 | 31.71% | 22 | 13 | 229 | 361 | 28 | 31.71% | 3.48% |
| xFuzz | 8 | 18 | 0 | 0 | 15 | 54.55% | 463 | 18 | 155 | 2 | 15 | 54.55% | 90.00% |
| Slither | 5 | 31 | 0 | 0 | 5 | 86.11% | 39 | 31 | 494 | 84 | 5 | 86.11% | 26.96% |
| Oyente | 5 | 28 | 0 | 0 | 8 | 77.78% | 39 | 28 | 575 | 3 | 8 | 77.78% | 90.32% |
| sFuzz | 5 | 18 | 0 | 0 | 18 | 50.00% | 39 | 18 | 574 | 4 | 18 | 50.00% | 81.82% |
| Confuzzius | 5 | 22 | 0 | 0 | 14 | 61.11% | 39 | 22 | 577 | 1 | 14 | 61.11% | 95.65% |

vulnerabilities vary among the tools, and reentrancy can be used as a standardized metric for one of the most common smart contract vulnerabilities. Second, reentrancy vulnerabilities have proven their impact in the real world, whereas many other vulnerabilities exist in theory.

## E. Dataset Collection

There are two challenges in constructing evaluation datasets. The first is caused by tool constraints. For example, GURU only allows manual data entry via its Web GUI interface, significantly restricting the number of evaluated smart contracts. Furthermore, different tools have varying input format requirements, resulting in huge complexity in dataset preparation. Second, our evaluation should reflect the tools' performance in a production environment. Thus, we select 41 smart contract instances with reentrancy vulnerabilities from the human-validated *ReentrancyStudy-Data* [57] repository.

We organize three Datasets. *Dataset 1* includes all 41 TP cases, specifically for testing GURU, which does not support batch analysis. *Dataset 2* comprises a broader selection of 653 contracts, including both TP and TN cases, allowing for a large-scale evaluation with tools like DLVA and baselines[1]. *Dataset 3* consists of real-world smart contracts from 5 reentrancy events from 2022 to 2024 to evaluate the tools' utility in real DeFi applications. By tailoring the dataset to the capabilities of the analysis tools, we ensure a robust and fair evaluation while maintaining high standards for evaluating research objectives.

## F. Recall and Precision

We compare three tools (i.e., GURU, DLVA and xFuzz) with four traditional tools on *dataset 1* and *dataset 2* regarding precision and recall metrics to evaluate their performance comprehensively. The results are shown in Table IV.

*Dataset 1* contains 41 contracts, GURU demonstrates the best performance, with a recall of 100%, while the recall of other tools ranges between 31.71% and 86.11%. This result indicates that GURU has excellent vulnerability detection capabilities on small datasets. However, we cannot assess its performance on larger datasets due to the lack of data for GURU on *dataset 2*.

[1]We download all the bytecodes of collected smart contracts through Etherscan and classify smart contracts based on their bytecode size.

TABLE V
EFFICIENCY METRICS FOR SIX SECURITY ANALYSIS TOOLS IN THE CASE OF DATASET 2. MCAT STANDS FOR MEAN CONTRACT ANALYSIS TIME, I.E. THE AVERAGE TIME REQUIRED TO ANALYSE A SMART CONTRACT

| Tool | Dataset 2 (size=653) | | | | |
|---|---|---|---|---|---|
| | Duration (sec) | TP | # of Contracts | TPT (sec) | MAT (sec) |
| DLVA | 540 | 13 | 631 | 41.54 | 0.86 |
| xFuzz | 15210 | 18 | 190 | 845.00 | 80.05 |
| Slither | 3771 | 31 | 614 | 121.65 | 6.14 |
| Oyente | 44099 | 28 | 614 | 1,574.96 | 71.82 |
| Confuzzius | 197505 | 22 | 614 | 8,977.50 | 321.67 |
| sFuzz | 73619 | 18 | 614 | 4,089.94 | 119.90 |

*Dataset 2* consists of 653 contracts, allowing for a more comprehensive examination of the performance of each tool. On this dataset, Confuzzius ranks first with a precision of 95.65%, closely followed by Oyente with a precision of 90.32%. Among the AI-powered tools, xFuzz performs best, achieving a precision of 90.00%, only slightly lower than Oyente. This suggests that xFuzz can control false positives nearly as well as top traditional tools. In contrast, DLVA's precision was only 3.48%, significantly lagging behind the other tools. xFuzz, while demonstrating precision comparable to top traditional tools at 90.00%, had significant analysis failures (463) on *dataset 2* due to its requirement for a specific smart contract compiler version, limiting its applicability. By contrast, traditional tools like Confuzzius outperform in precision on *dataset 2* at 95.65%, indicating strong FP control, with Slither notes for high recall on *dataset 1* but lower precision on *dataset 2*. Oyente and sFuzz had balanced precision but required recall improvements.

Overall, this assessment indicates that the existing AI-based smart contract vulnerability detection tools have mixed performance in terms of precision and recall and have yet to be able to surpass traditional tools completely. However, xFuzz has shown the potential of AI tools, with its precision already approaching the level of top traditional tools.

## G. Efficiency

Table V shows the efficiency results of these tools. DLVA, performs poorly in the recall and precision experiment, however, demonstrates exceptional performance in execution efficiency. It has the lowest *TPT* of 41.54 seconds and *MAT* of 0.86 seconds. These results indicate that DLVA is quick and efficient in identifying real vulnerabilities. One reason for DLVA's high efficiency is its ML-based design philosophy of directly identifying smart contract vulnerabilities.

Confuzzius performs the worst on both indicators, with a *TPT* of 8,977.50 seconds and an *MAT* of 321.67 seconds, far exceeding the other tools. One reason is that Confuzzius, as a fuzz testing tool that combines symbolic execution, involves complex or computationally intensive analysis overhead. Oyente and sFuzz perform poorly on *TPT*, with 1,574.96 seconds and 4,089.94 seconds, respectively. The efficiency issues are mainly caused by more in-depth data-flow analysis when dealing with certain types of smart contracts, which consumes more time when uncovering real vulnerabilities.

Slither displays a balanced efficiency with a moderate *TPT* of 121.65 seconds and a lower *MAT* of 6.14 seconds. In contrast, even though xFuzz performs poorly on the *MAT* with 80.05 seconds, its *TPT* of 845.00 seconds is much lower than that of Confuzzius and sFuzz. This may benefit from the path filtering algorithm that xFuzz integrates with AI technology, reducing irrelevant path exploration during the analysis.

### *H. Utility*

As shown in Table VI, both GURU and DLVA correctly detect 3 vulnerabilities in the 5 DeFi applications, and the detection results of these two tools only have 1 overlap case, which suggests that the different AI-powered Tools may be mutually beneficial.

However, xFuzz performs relatively poorly in this evaluation and does not successfully detect any vulnerability. The CFG module of xFuzz cannot handle the remaining 4 smart contracts[2] except for VOLTAGE, making the fuzz testing phase impossible.

The result indicates that DLVA and GURU have greater utility in detecting vulnerabilities in real DeFi applications because they show ideal scalability. Although xFuzz shows excellent performance on *dataset 1* and *dataset 2*, its reliance on richer code features limits its utility when dealing with higher complexity real-world DeFi applications.

### *I. Limitation Identification*

We identify some limitations and challenges of evolution, reflecting the shortcomings of the smart contract's current AI-powered security analysis. We analyze these limitations into four main types, as shown in Table VII.

**The Special Input Form Requirements**. DLVA requires users to provide the blockchain address of the smart contract and the corresponding bytecode as input, increasing analysts' data preparation workload and limiting the tool's flexibility.

**The Exception Handling Defects.** Although DLVA can detect and report exceptional inputs, it cannot effectively isolate exceptional data from the normal analysis process, interrupting the entire batch analysis process once an exception occurs. This will become a bottleneck when facing large-scale heterogeneous data.

**The Limited Ability for Large-scale Analysis.** GURU, as a web GUI-based tool, currently lacks a command-line interface for efficient batch process. As a result, it is difficult to apply it to large-scale security analysis of smart contracts.

**The Lack of Formatted Output Options.** Although xFuzz provides intuitive command-line output for users, unfortunately, it does not offer formatted output options, which brings certain difficulties to subsequent data statistics and analysis.

To sum up, the aforementioned limitations reflect that the current AI-powered security analysis tools still have large room for practicality, usability, and robustness improvement. Analyzing these shortcomings can provide valuable guidance for optimizing and improving subsequent tools.

### *J. Summary*

The evaluation results of xFuzz, DLVA, and GURU reveal the diverse strengths and weaknesses of AI-powered smart contract security analysis. xFuzz showcases the promise of integrating AI technology with traditional smart contract security analysis. It achieves a precision level comparable to top traditional tools. This suggests that AI technology can effectively guide path exploration, leading to more accurate vulnerability detection. However, xFuzz's efficiency, particularly in *MAT*, leaves much to be desired. Its reliance on a specific compiler version limits flexibility and utility to diverse smart contract environments and real-world DeFi applications.

DLVA, despite its poor precision and recall, exhibits remarkable efficiency in both *TPT* and *MAT*. This indicates that its end-to-end classification method, which directly identifies vulnerabilities in the smart contract, has the potential to streamline the analysis process significantly. GURU's performance on the small dataset suggests that its advanced GNN architecture has the potential to capture intricate patterns and dependencies in smart contract code, enabling effective vulnerability detection.

## VI. Research Opportunities

The above investigation and evaluation indicate the opportunities for AI-powered smart contract security analysis to promote analysis generalization and interpretability.

**Generalization.** Generalizing vulnerability detection models across diverse and evolving smart contract ecosystems is a key requirement mentioned by many prior research. The analysis generalization enables users to extend and migrate a solution for one security issue to another. To achieve this, there are two directions for developing stable models:

- *Expanding Training Datasets.* Large and diverse smart contract datasets are crucial for models to learn more patterns. Thus, various information, such as source code, bytecode, high-level semantic information, etc., has to be collected for model training.

[2]Specifically, LENDF.ME's analysis times out during the CFG construction process. In contrast, the remaining 3 smart contracts seem to be due to an exception within the CFG module.

TABLE VI

UTILITY EVALUATION RESULTS FOR THREE TOOLS FOR 6 REENTRANCY EVENTS FROM 2021 TO 2024.

| DeFi Application | Attack Date | Vulnerable Contract Address | Platform | Solc Version | xFuzz | GURU | DLVA |
|---|---|---|---|---|---|---|---|
| LENDF.ME | 2020-04 | 0x0eee3e3828a45f7601d5f54bf49... | Ethereum | 0.4.25 | - | - | ✔ |
| VOLTAGE | 2022-03 | 0xa722c13135930332eb3d749b2f0... | Fuse | 0.4.24 | - | ✔ | - |
| SUSHIBAR | 2022-10 | 0x2321537fd8ef4644bacdceec54e... | Ethereum | 0.8.16 | - | ✔ | ✔ |
| XSURGE | 2021-08 | 0xe1e1aa58983f6b8ee8e4ecd206c... | Binance | 0.8.5 | - | ✔ | - |
| NEBULA | 2024-01 | 0x5499178919c79086fd580d6c5f3... | Ethereum | 0.8.18 | - | - | ✔ |

TABLE VII

LIST OF TOOLS LIMITATION THAT APPEAR IN THE EVALUATION PROCESS

| Limitation Type | Tool | | |
|---|---|---|---|
| | GURU | DLVA | xFuzz |
| Special input form requirements | - | ✔ | - |
| Exception handling defects | - | ✔ | - |
| Limited ability for large-scale analysis | ✔ | - | - |
| Lack of formatted output options | - | - | ✔ |

- *Improving Supervised Learning Techniques.* To overcome the problem of large discrepancies in data labeling between different research in handling smart contract data, subsequent research can focus on weakly supervised learning techniques. Research into weakly supervised methods can provide ways to leverage large amounts of unlabeled data, thereby enhancing the model's ability to generalize from limited examples.

**Interpretability.** AI-powered smart contract security analysis aims to help developers create safer smart contracts. Therefore, creating developer-friendly tools with better interpretability is important. In other words, AI-powered smart contract security analysis should present detected security issues in a more accessible way for developers.

## VII. CONCLUSION

In this paper, we comprehensively study existing AI-powered smart contract security analysis from 2018 to 2024. We analyze 27 representative studies and conduct empirical evaluations using three open-source tools. We find that these tools still need improvement in precision, tool usability, and practicality compared with traditional smart contract security analysis tools. There are several directions to extend this work. First, the scope of the literature search can be extended to cover more databases, conference papers, and industry reports. Second, text mining and knowledge graph technologies can be utilized to automate the assessment of literature quality, reducing bias caused by subjective judgment.

## REFERENCES

[1] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "SoK: Decentralized Finance (DeFi) Attacks," in *the 44th IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2444–2461.

[2] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," *Technological Forecasting and Social Change*, vol. 144, pp. 1–11, 2019.

[3] D. Sheridan, J. Harris, F. Wear, J. Cowell Jr, E. Wong, and A. Yazdinejad, "Web3 challenges and opportunities for the market," *arXiv preprint arXiv:2209.02446*, 2022.

[4] B. Sriman and S. G. Kumar, "Decentralized finance (DeFi): the future of finance and DeFi application for Ethereum blockchain based finance market," in *the 2nd International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*. IEEE, 2022, pp. 1–9.

[5] W. W. Ding, X. Liang, J. Hou, G. Wang, Y. Yuan, J. Li, and F.-Y. Wang, "Parallel governance for decentralized autonomous organizations enabled by blockchain and smart contracts," in *the 1st IEEE International Conference on Digital Twins and Parallel Intelligence (DTPI)*. IEEE, 2021, pp. 1–4.

[6] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, 2018, pp. 653–663.

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

[8] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 254–269.

[9] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in Ethereum smart contracts," in *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*, 2018, pp. 664–676.

[10] 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 42nd IEEE/ACM International Conference on Software Engineering (ICSE)*, 2020, pp. 778–788.

[11] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, "Confuzzius: A data dependency-aware hybrid fuzzer for smart contracts," in *the 42nd IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 103–119.

[12] M. Zhang, X. Zhang, Y. Zhang, and Z. Lin, "TXSPECTOR: Uncovering attacks in Ethereum from transactions," in *the 29th USENIX Security Symposium (USENIX Security)*, 2020, pp. 2775–2792.

[13] L. Su, X. Shen, X. Du, X. Liao, X. Wang, L. Xing, and B. Liu, "Evil under the sun: Understanding and discovering attacks on Ethereum decentralized applications," in *the 30th USENIX Security Symposium (USENIX Security)*, 2021, pp. 1307–1324.

[14] W. J.-W. Tann, X. J. Han, S. S. Gupta, and Y.-S. Ong, "Towards safer smart contracts: A sequence learning approach to detecting security threats," *arXiv preprint arXiv:1811.06632*, 2018.

[15] S. J. Wang, K. Pei, and J. Yang, "SMARTINV: Multimodal Learning for Smart Contract Invariant Inference," in *the 45th IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 126–126.

[16] Y. Gai, L. Zhou, K. Qin, D. Song, and A. Gervais, "Blockchain Large Language Models," *arXiv preprint arXiv:2304.12749*, 2023.

[17] Y. Xue, J. Ye, W. Zhang, J. Sun, L. Ma, H. Wang, and J. Zhao, "xFuzz: Machine learning guided cross-contract fuzzing," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[18] T. Abdelaziz and A. Hobor, "Smart learning to find dumb contracts," in *the 32nd USENIX Security Symposium (USENIX Security)*, 2023, pp. 1775–1792.

[19] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan, and L. Jiang, "MANDO-GURU: Vulnerability Detection for Smart Contract Source Code By Heterogeneous Graph Embeddings," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and*

*Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022, New York, NY, USA, 11 2022, pp. 1736–1740.

[20] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[21] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "SmartInspect: Solidity smart contract inspector," in *the 1st International workshop on blockchain oriented software engineering (IWBOSE)*. IEEE, 2018, pp. 9–18.

[22] S. Cui, G. Zhao, Y. Gao, T. Tavu, and J. Huang, "Vrust: Automated vulnerability detection for Solana smart contracts," in *the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 639–652.

[23] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[24] V. Buterin, "Chain interoperability," *R3 research paper*, vol. 9, pp. 1–25, 2016.

[25] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: Finding reentrancy bugs in smart contracts," in *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE): Companion Proceeedings*, 2018, pp. 65–68.

[26] Y. Xue, M. Ma, Y. Lin, Y. Sui, J. Ye, and T. Peng, "Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 1029–1040.

[27] Y. Chinen, N. Yanai, J. P. Cruz, and S. Okamura, "Ra: Hunting for Reentrancy attacks in Ethereum smart contracts via static analysis," in *the 3rd IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 327–336.

[28] A. Ghaleb, J. Rubin, and K. Pattabiraman, "Achecker: Statically detecting smart contract access control vulnerabilities," in *the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 945–956.

[29] Y. Fang, D. Wu, X. Yi, S. Wang, Y. Chen, M. Chen, Y. Liu, and L. Jiang, "Beyond "Protected" and "Private": An Empirical Security Analysis of Custom Function Modifiers in Smart Contracts," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 1157–1168.

[30] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *the 30th USENIX Security Symposium (USENIX Security)*, 2021, pp. 1325–1341.

[31] P. Praitheeshan, L. Pan, X. Zheng, A. Jolfaei, and R. Doss, "Solguard: Preventing external call issues in smart contract-based multi-agent robotic systems," *Information Sciences*, vol. 579, pp. 150–166, 2021.

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

[33] P. Qian, Z. Liu, Q. He, B. Huang, D. Tian, and X. Wang, "Smart contract vulnerability detection technique: A survey," *arXiv preprint arXiv:2209.05872*, 2022.

[34] T. Hu, Z. Li, B. Li, and Q. Bao, "Contractual security and privacy security of smart contract: a system mapping study," *Chinese Journal of Computers*, vol. 44, no. 12, pp. 2485–2514, 2021.

[35] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting Ponzi schemes on Ethereum: Towards healthier blockchain technology," in *Proceedings of the 27th World Wide Web Conference (WWW)*, 2018, pp. 1409–1418.

[36] H. Liu, C. Liu, W. Zhao, Y. Jiang, and J. Sun, "S-gram: towards semantic-aware security auditing for Ethereum smart contracts," in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 814–819.

[37] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "Contractward: Automated vulnerability detection models for Ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2020.

[38] Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Grundy, "Checking smart contracts with structural code embedding," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2874–2891, 2020.

[39] 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 29th International Conference on International Joint Conferences on Artificial Intelligence (IJCAI)*, 2021, pp. 3283–3290.

[40] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1296–1310, 2021.

[41] S. So, S. Hong, and H. Oh, "SmarTest: Effectively hunting vulnerable transaction sequences in smart contracts through language Model-Guided symbolic execution," in *the 30th USENIX Security Symposium (USENIX Security)*, 2021, pp. 1361–1378.

[42] J. Huang, S. Han, W. You, W. Shi, B. Liang, J. Wu, and Y. Wu, "Hunting vulnerable smart contracts via graph embedding based bytecode matching," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2144–2156, 2021.

[43] N. He, R. Zhang, H. Wang, L. Wu, X. Luo, Y. Guo, T. Yu, and X. Jiang, "EOSAFE: Security analysis of EOSIO smart contracts," in *the 30th USENIX Security Symposium (USENIX Security)*, 2021, pp. 1271–1288.

[44] X. Hu, Z. Gao, X. Xia, D. Lo, and X. Yang, "Automating user notice generation for smart contract functions," in *the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 5–17.

[45] S. Fan, S. Fu, H. Xu, and X. Cheng, "Al-SPSD: Anti-leakage smart Ponzi schemes detection in blockchain," *Information Processing & Management*, vol. 58, no. 4, p. 102587, 2021.

[46] J. Liu, Y. Chen, B. Tan, I. Dillig, and Y. Feng, "Learning contract invariants using reinforcement learning," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022, pp. 1–11.

[47] J. Su, H.-N. Dai, L. Zhao, Z. Zheng, and X. Luo, "Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022, pp. 1–12.

[48] Y. Pan, Z. Xu, L. T. Li, Y. Yang, and M. Zhang, "Automated generation of security-centric descriptions for smart contract bytecode," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023, pp. 1244–1256.

[49] W. Jie, Q. Chen, J. Wang, A. S. V. Koe, J. Li, P. Huang, Y. Wu, and Y. Wang, "A novel extended multimodal ai framework towards vulnerability detection in smart contracts," *Information Sciences*, vol. 636, p. 118907, 2023.

[50] Z. Zheng, W. Chen, Z. Zhong, Z. Chen, and Y. Lu, "Securing the Ethereum from smart Ponzi schemes: Identification using static features," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–28, 2023.

[51] A. Storhaug, J. Li, and T. Hu, "Efficient avoidance of vulnerabilities in auto-completed smart contract code using vulnerability-constrained decoding," in *the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 683–693.

[52] S. Hu, T. Huang, F. İlhan, S. F. Tekin, and L. Liu, "Large Language Model-powered smart contract vulnerability detection: New perspectives," *arXiv preprint arXiv:2310.01152*, 2023.

[53] C. Chen, J. Su, J. Chen, Y. Wang, T. Bi, Y. Wang, X. Lin, T. Chen, and Z. Zheng, "When ChatGPT meets smart contract vulnerability detection: How far are we?" *arXiv preprint arXiv:2309.05520*, 2023.

[54] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "When GPT meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in GPTScan," *arXiv preprint arXiv:2308.03314*, 2023.

[55] R. Liang, J. Chen, K. He, Y. Wu, G. Deng, R. Du, and C. Wu, "PonziGuard: Detecting Ponzi Schemes on Ethereum with Contract Runtime Behavior Graph (CRBG)," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024, pp. 1–12.

[56] M. Di Angelo, T. Durieux, J. F. Ferreira, and G. Salzer, "Smartbugs 2.0: An execution framework for weakness detection in Ethereum smart contracts," in *the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 2102–2105.

[57] Z. Zheng, N. Zhang, J. Su, Z. Zhong, M. Ye, and J. Chen, "Turn the rudder: A beacon of reentrancy detection for smart contracts on Ethereum," in *the 45th IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 295–306.