electronics

MDPI

Review

Enhancing Smart-Contract Security through Machine Learning: A Survey of Approaches and Techniques

Fan Jiang [1,2,3], Kailin Chao [1,2,3], Jianmao Xiao [1,2,3,*], Qinghua Liu [1,2,3], Keyang Gu [1,2,3], Junyi Wu [1,2,3] and Yuanlong Cao [1,2,3]

[1] School of Software, Jiangxi Normal University, Nanchang 330022, China; jiangfan@jxnu.edu.cn (F.J.)
[2] Jiangxi Provincial Engineering Research Center of Blockchain Data Security and Governance, Nanchang 330022, China
[3] Management Science and Engineering Center, Jiangxi Normal University, Nanchang 330022, China
* Correspondence: jm_xiao@jxnu.edu.cn

Abstract: As blockchain technology continues to advance, smart contracts, a core component, have increasingly garnered widespread attention. Nevertheless, security concerns associated with smart contracts have become more prominent. Although machine-learning techniques have demonstrated potential in the field of smart-contract security detection, there is still a lack of comprehensive review studies. To address this research gap, this paper innovatively presents a comprehensive investigation of smart-contract vulnerability detection based on machine learning. First, we elucidate common types of smart-contract vulnerabilities and the background of formalized vulnerability detection tools. Subsequently, we conduct an in-depth study and analysis of machine-learning techniques. Next, we collect, screen, and comparatively analyze existing machine-learning-based smart-contract vulnerability detection tools. Finally, we summarize the findings and offer feasible insights into this domain.

Keywords: machine learning; safety; smart contract; vulnerability detection; survey

Citation: Jiang, F.; Chao, K.; Xiao, J.; Liu, Q.; Gu, K.; Wu, J.; Cao, Y. Enhancing Smart-Contract Security through Machine Learning: A Survey of Approaches and Techniques. *Electronics* **2023**, *12*, 2046. https://doi.org/10.3390/electronics12092046

Academic Editor: Mehdi Sookhak

Received: 30 March 2023
Revised: 13 April 2023
Accepted: 18 April 2023
Published: 28 April 2023

1. Introduction

The introduction of blockchain technology as the foundation for Bitcoin by Satoshi Nakamoto in 2008 [1] marked the beginning of a new era in decentralized data management frameworks with the potential to revolutionize traditional industries [2]. Over the past decade, blockchain technology has experienced rapid development not only experienced in the financial sector but also in various other fields, such as supply chain management [3], the Internet of Things (IoT) [4,5], transportation [6], retail [7], healthcare [8,9], gaming [10], and communication [11,12], among others. The impact of blockchain technology had progressively driven global economic growth by 2020, and it is projected to contribute 1.76 trillion dollars to the global economy by 2030 by increasing traceability and trust levels [13].

As blockchain technology progresses and permeates various domains, considering it merely a distributed ledger no longer captures its true essence. The tremendous potential of blockchain technology across diverse sectors has garnered significant attention and heightened expectations. Nonetheless, to realize these expectations, several challenges must be addressed. First, a sole consensus mechanism faces difficulties in ensuring high security and trustworthiness in a decentralized transaction verification network within a distributed environment where trust is not absolute. Second, the scalability challenge of blockchain technology has considerably hindered its integration and growth across various industries and fields.

In this context, Ethereum, as a groundbreaking blockchain platform [14], provides a viable solution to these challenges through the implementation of smart-contract technology. Smart contracts are self-executing, blockchain-based computer protocols that

facilitate highly secure and trustworthy transactions in a decentralized environment. Consequently, smart contracts have been employed in a wide range of applications, including IoT-based smart contracts [15], trusted database systems using smart contracts [16], and smart-contract-based medical data consent models [9,17], among others. Given the significant sums of money involved in various smart-contract applications, ensuring their security is of paramount importance. As a result, numerous researchers and engineers have devoted their efforts to examining and enhancing the security of smart contracts, aiming to unlock their full potential across a diverse array of use cases.

An effective approach to smart-contract security detection is the application of machine-learning techniques. Over the past few decades, the field of machine learning has made significant progress and has become one of the key areas in modern computer science. The primary goal of machine learning is to develop algorithms capable of learning autonomously from data, enabling them to make predictions, classifications, or decisions when faced with new data.

Existing machine-learning-based smart-contract security detection has achieved a certain scale. However, in the current literature, comprehensive reviews on the application of machine learning in smart-contract security detection are relatively rare. To address this gap, we have decided to write a review paper, aiming to outline the current state, challenges, and future development trends of machine-learning techniques in smart-contract security detection, providing a comprehensive and systematic reference for both academics and practitioners. Our review is guided by the following core research questions:

RQ1: What machine-learning methods are suitable for smart-contract security detection, and what are the advantages and limitations of these methods in vulnerability detection?

RQ2: What existing approaches apply machine learning to smart-contract security detection, and how do these methods perform?

RQ3: Future research directions and challenges: How can machine-learning methods be combined with other security analysis techniques to further improve the performance of smart-contract vulnerability detection?

In Section 2, we will introduce related review studies. In Section 3, we will introduce the research background of the article, including common types of smart-contract vulnerabilities and existing non-machine-learning tools for smart-contract vulnerability detection. In Section 4 will delve into machine-learning techniques applicable to smart-contract security detection. In Section 5 will provide an overview and analysis of related work that has been practically applied in smart-contract security detection. In Section 6, we will explore in-depth the three research questions posed by this study. Finally, in Section 7, we will summarize the main findings and conclusions of this paper.

2. Related Work

In this chapter, we have conducted an in-depth investigation of recent review literature in the field of smart-contract vulnerability detection, as well as reviews related to vulnerability detection based on machine learning. However, during this process, we noticed a concerning phenomenon: while machine learning has made significant progress in smart-contract vulnerability detection, there is still a noticeable gap in the review literature specifically addressing machine-learning-based smart-contract vulnerability detection. Therefore, the aim of this paper is to fill this void, providing researchers and developers with a more comprehensive and systematic theoretical framework and practical reference in this domain.

Atzei et al. [18] conducted a research investigation on Ethereum smart-contract attacks. Their study focused on various attack types and case studies, as well as ways to improve the security of smart contracts. Their work centered on analyzing different attack strategies, thus providing developers with practical advice and tools on how to defend against these attacks. Chen et al. [19] carried out a literature review on Ethereum system security, with a particular emphasis on vulnerabilities, attacks, and defenses. Overall, their research pro-

vided an in-depth understanding of the security challenges faced by the Ethereum platform and proposed several countermeasures. Their work thoroughly assessed the security of the Ethereum ecosystem, suggested future research directions, and highlighted the importance of defensive measures. Liu et al. [20] surveyed and summarized the security verification of blockchain smart contracts. Their research work mainly focused on security verification methods and techniques for smart contracts. Their work was devoted to exploring various verification techniques rather than being limited to a single technology (such as formal verification), thus offering a comprehensive overview of various smart-contract security verification methods. Kabla et al. [21] conducted a comprehensive investigation on the applicability of Intrusion Detection Systems (IDS) for Ethereum attacks. This study examined the effectiveness of various IDS technologies in addressing security issues and detecting potential threats on the Ethereum blockchain. The investigation not only surveyed existing IDS methods but also provided new insights into the challenges and opportunities for enhancing intrusion-detection capabilities within the Ethereum ecosystem. Furthermore, the authors developed a multi-dimensional classification framework to assess and compare different IDS technologies, deepening the understanding of their strengths and weaknesses. Rameder et al. [22] performed a literature review on the automated vulnerability analysis of Ethereum smart contracts. This review mainly explored various automated vulnerability analysis techniques for smart contracts on the Ethereum platform. The review focused more on the application and potential of automated analysis methods in the smart-contract security domain and proposed a multi-dimensional classification framework to analyze and evaluate different vulnerability analysis techniques. Kushwaha et al. [23] conducted a systematic survey of security vulnerabilities in Ethereum blockchain smart contracts. The study delved into various security vulnerabilities plaguing Ethereum smart contracts and discussed techniques for identifying and mitigating these issues. The novelty of this work lies in the systematic approach taken to examine security vulnerabilities and their impacts while also identifying gaps in the current state of research and proposing future directions for enhancing the security of smart contracts on the Ethereum platform. Krichen et al. [24] innovatively investigate the application of formal methods in the specification and verification of smart contracts, aiming to reduce the risk of failures and minimize costs. Concurrently, it proposes future research directions, such as lowering verification costs, integrating various mathematical concepts, and enhancing the accessibility and reusability of formal methods. These insights provide novel perspectives and guidance for the smart-contract domain. Miller et al. [25] innovatively propose utilizing formal methods for systematic auditing and verification of smart contracts to ensure their security. Simultaneously, they examine the platforms, high-profile vulnerabilities, and existing analysis tools within the smart-contract ecosystem and identify the research challenges faced by formal methods and program analysis applied to smart contracts.

Additionally, we conducted a survey of review articles on vulnerability-detection models based on machine-learning techniques. Ahmed et al. [26] published a machine-learning review on software vulnerability detection at the 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM) in 2022. This review primarily explored the application and outcomes of machine-learning methods in the field of software vulnerability detection. Unlike other reviews, this article delved into the practical application and effectiveness of various machine-learning techniques in software vulnerability detection and proposed a classification framework to illustrate the types of technologies employed by each method. Pan et al. [27] carried out a literature review on hardware vulnerability analysis using machine learning. This review mainly investigated the application and results of machine-learning methods in the field of hardware vulnerability analysis. They thoroughly examined the practical application and effectiveness of various machine-learning techniques in hardware vulnerability analysis and proposed a classification framework to describe the types of technologies adopted by each method. Zeng et al. [28] conducted a survey review on software vulnerability analysis using deep-learning techniques. This review primarily discussed the application and effectiveness of

deep-learning methods in the field of software vulnerability analysis and discovery. This research did not focus on any specific technology but rather discussed the target technology as a dimension of the classification framework. Lin et al. [29] performed a literature review on software vulnerability detection using deep learning techniques. This review provided an in-depth discussion of the application and outcomes of deep neural network methods in software vulnerability detection, offering insightful perspectives and future research directions. Additionally, this review paid particular attention to the verification aspect, considering not only security-related issues but also functional problems.

3. Background

3.1. Classification of Smart-Contract Vulnerabilities

3.1.1. Reentrancy Attack

Reentrancy attacks are a common vulnerability in smart contracts, where an attacker repeatedly calls functions during contract interaction, allowing them to steal assets or disrupt contract logic. To prevent such attacks, contract developers should follow the "check-effects-interactions" principle, ensuring state updates are completed before interacting with external contracts. Employing locking mechanisms can also help avoid reentrancy. Careful code review, testing, and adherence to programming best practices can help protect against reentrancy attacks and other security vulnerabilities.

Figure 1 illustrates a simplified bank smart contract with a reentrancy attack vulnerability and an attacker's contract designed to exploit this vulnerability. In this banking system, users can deposit funds into their bank accounts (using the *deposit*() function), check their balances (using the *getBalance*() function), and withdraw funds (using the *withdraw*() function).

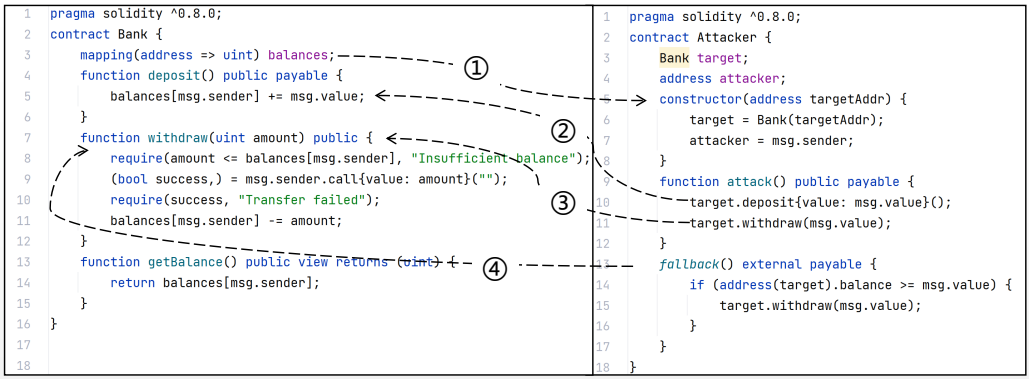

Figure 1. Reentrancy Attack.

However, there is a flaw within the *withdraw*() function that attackers can exploit for a reentrancy attack. When a user calls the *withdraw*() function, the contract sends a specified amount of Ether to the user's address. If the user's address is a contract address and that contract implements a $fallback()$ function, this $fallback()$ function could trigger a new *withdraw*() call, leading to a reentrancy attack. In the example, we outline the attacker's process: the attacker creates a contract called Attacker, passing the target Bank contract's address and their own address in the constructor. In the *attack*() function, the attacker first deposits a certain amount of Ether into the target Bank contract's account, then calls the *withdraw*() function. In the $fallback()$ function, if the attacker receives enough Ether, they transfer the Ether to the target contract and call the *withdraw*() function again. This causes the target contract to repeatedly execute the *withdraw*() function, resulting in multiple withdrawal operations and theft of the contract's assets. Therefore, by implementing a malicious contract, the attacker exploits the reentrancy attack to bypass the security mechanisms in the target contract and successfully steal its assets.

To prevent such attacks, developers should be cautious about reentrancy vulnerabilities when calling functions from other contracts and carefully consider security during

contract design and implementation. Researchers and developers can use various static and dynamic analysis tools to detect and fix vulnerabilities during the vulnerability discovery and remediation process. Additionally, contract audits and security assessments are crucial steps to ensure contract safety.

3.1.2. Integer Overflow and Underflow

Smart-contract integer overflow and underflow vulnerabilities are common issues in smart contracts. An integer overflow or underflow occurs when an integer variable's value exceeds or falls below the maximum or minimum value that its data type can represent. In smart contracts, integer overflow or underflow can lead to calculation errors or unpredictable behavior, potentially compromising the contract's security. To address integer overflow and underflow problems, smart-contract developers typically use the SafeMath library, which offers a set of secure mathematical operations to prevent overflow and underflow.

In Figure 2, we present a simplified bank system smart contract and an example of an attack contract targeting this smart contract. In the bank contract on the left, users can deposit funds into the bank account (using the $deposit()$ function), check their balance (using the $getBalance()$ function), and withdraw funds (using the $withdraw()$ function). However, the balances variable in the contract is of uint type, and if a user deposits more than 2^{255} ether, it may cause an integer overflow in the balances variable, resulting in a significantly reduced user balance. Conversely, if a user attempts to withdraw an amount greater than $balances[msg.sender]$, it may lead to integer underflow, turning the user's balance into a large value or even the maximum value in the contract. The attack contract shown on the right exploits this situation to attack the bank system. During the attack, the attacker first creates a contract called Attacker and passes the target contract Bank's address in the constructor. Then, in the $attack()$ function, the attacker deposits 2^{255} ether into the target contract Bank's account and subsequently calls the $withdraw()$ function to withdraw 1 ether. Due to $balances[msg.sender]$ becoming a large value, withdrawing 1 ether causes $balances[msg.sender]$ to transform into a small value, leading to an integer underflow vulnerability. The attacker uses the $getBalance()$ function to check the balance and stores the result in a public variable called overflow, allowing other users to view it.

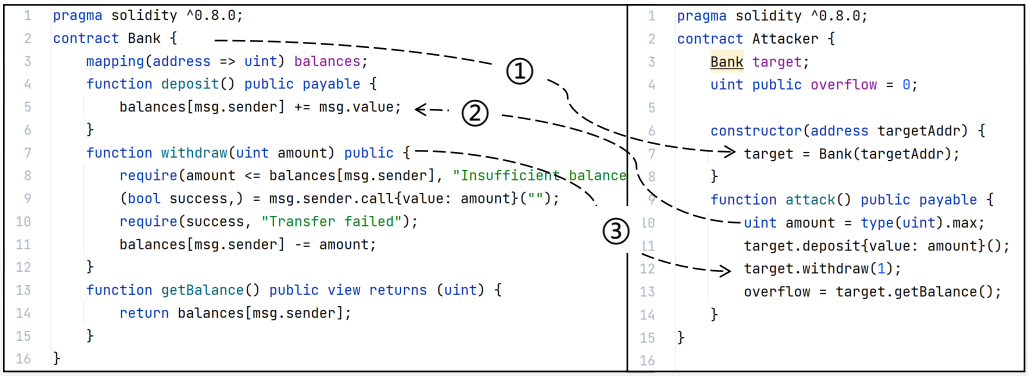

Figure 2. Integer Overflow and Underflow.

To prevent such attacks, developers should be cautious when using integer variables to avoid overflow or underflow, carefully considering security in contract design and implementation. In detecting and fixing vulnerabilities, researchers and developers can utilize various static and dynamic analysis tools to help identify and resolve issues. Additionally, contract auditing and security assessments are crucial steps in ensuring contract safety.

3.1.3. Uninitialized Storage Pointer

Uninitialized Storage Pointers in smart contracts represent a class of easily overlooked but potentially dangerous security vulnerabilities. These vulnerabilities typically occur

when developers use storage pointers within smart contracts without initializing them, or when initialization is incomplete. Since storage pointers directly point to the storage space of a smart contract, attackers can exploit these pointers to access and modify data in the storage area, leading to incorrect reading, writing, or alteration of contract data.

In Figure 3, we present an example of a Voting System Smart Contract with an uninitialized storage pointer vulnerability and an attack contract targeting it. In the voting system, users can call the *vote*() function to cast their votes and the *getVotes*() function to query the voting results. However, the votes variable in the contract is a mapping type; if the attacker does not initialize it before calling the *getVotes*() function, an uninitialized storage pointer vulnerability may arise. Attackers can exploit this vulnerability to access and modify data in the storage area, thereby tampering with the voting results. In the attack contract shown on the right, the attacker creates a contract named Attacker and passes the address of the target Voting contract in the constructor. In the *attack*() function, the attacker first calls the *getVotes*() function, passing their own address as a parameter. As the attacker's address has not been voted for, the value of *msg.sender* is 0, leading to the uninitialized storage pointer vulnerability. Next, the attacker calls the *vote*() function to cast their vote, causing the value of votes *msg.sender* to be set to 1, successfully manipulating the voting results.

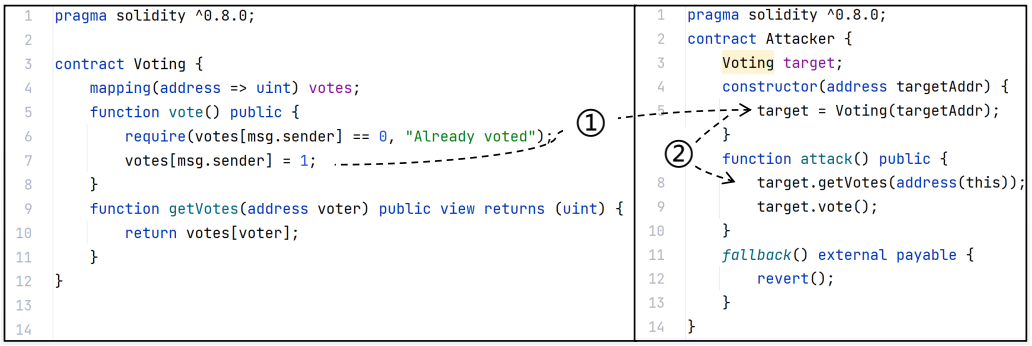

Figure 3. Uninitialized Storage Pointer.

To prevent such attacks, developers should ensure proper initialization of storage pointers and carefully consider security during the design and implementation of smart contracts. In detecting and fixing vulnerabilities, researchers and developers can employ various static and dynamic analysis tools to help identify and remediate issues. Additionally, auditing and security evaluations of contracts are vital steps in ensuring contract safety.

3.1.4. Access Control Vulnerability

Access Control Vulnerabilities refer to situations where smart contracts fail to correctly implement permission control, allowing attackers to carry out unauthorized operations. In general, a smart contract should verify a user's identity before allowing certain actions. However, if the contract fails to properly implement authentication, attackers can bypass these measures by impersonating authorized users or, through other means, executing unauthorized operations. Such vulnerabilities can lead to severe consequences, such as asset theft, contract tampering, or contract shutdown. Implementing proper access control in smart contracts is critical. Typically, functions within a contract should be limited to the contract owner or specified users. To enforce access control, contracts often use the *require*() statement in Solidity to check the caller's permissions. When checking permissions, contracts should adhere to the principle of least privilege, granting callers the minimum necessary permissions to execute an operation.

In Figure 4, we present an example of a smart contract with an access control vulnerability, along with an attacker's contract that exploits the vulnerability and the attack sequence. In this example, only the contract creator (owner) can execute the *doSomething*()

function, and others are not permitted. The contract sets the creator as the owner of its constructor. The *doSomething*() function uses the *require*() statement to check if the caller is the contract creator, and if not, the function execution fails and returns an error message. However, if someone obtains the private key of the contract creator, they can impersonate the creator to carry out an authorized user elevation attack and execute the *doSomething*() function. Therefore, even if the contract has implemented authentication, attackers can still bypass it if the private key is leaked or if other vulnerabilities are present.

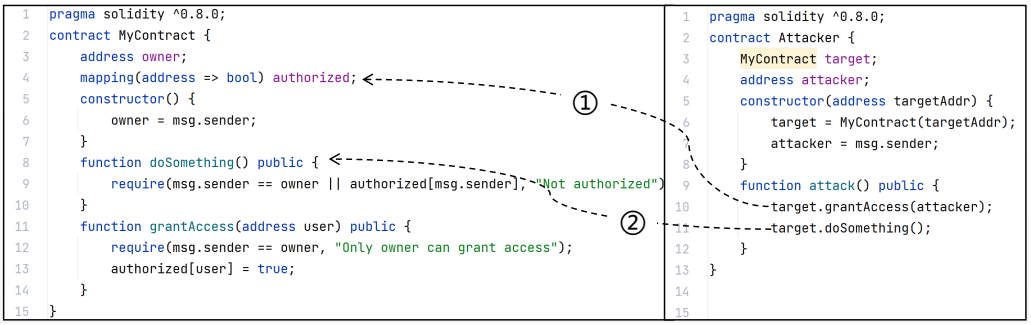

Figure 4. Access Control Vulnerability.

Hence, implementing proper access control in smart contracts is of utmost importance. Researchers should focus on concurrency control, role and permission design, and implementation, including aspects such as authentication and access control, to ensure the security of smart contracts.

3.1.5. Front-End Runtime Error Vulnerability

Front-end runtime error vulnerabilities often occur in the parts of smart contracts that interact with user interfaces. As smart contracts run within front-end applications, various errors may arise, including logic errors, input validation errors, and exceptional cases. These errors can lead to smart-contract execution failures or unpredictable behavior, ultimately compromising the contract's security.

Figure 5 is a voting system example; we create a voting system smart contract called "Voting," where users can call the *vote*() function to vote and the *getVotes*() function to query the voting results. However, the *vote*() function in this contract has a logic error: if a user votes multiple times, an exception is triggered, causing execution to fail. This may lead to the contract not executing correctly or exhibiting unpredictable behavior. Consequently, in the attack contract, the attacker exploits this function vulnerability by creating a contract called "Attacker" and passing the target contract Voting's address in the constructor. In the *attack*() function, the attacker calls the *vote*() function twice, causing the contract's execution to fail and potentially leading to unpredictable behavior.

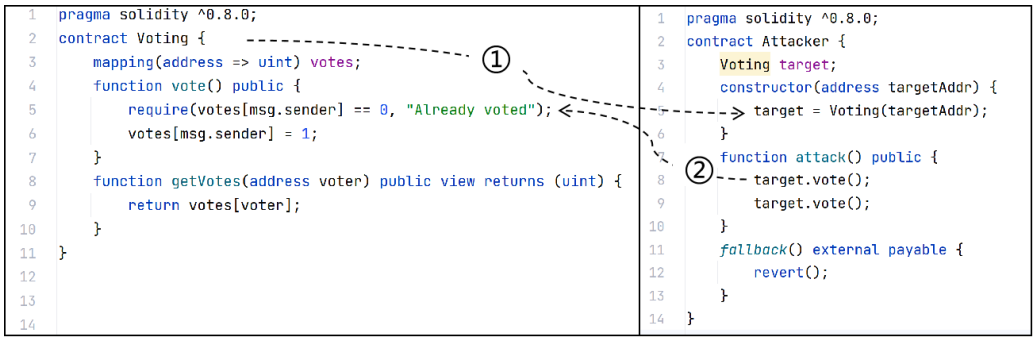

Figure 5. Front-end Runtime Error Vulnerability.

To prevent such attacks, developers should carefully consider the security of smart-contract front-end applications, including input validation, exception handling, and error

handling. When designing and implementing contracts, developers should fully consider potential error scenarios and adopt appropriate security measures to ensure the contract's correctness and safety. Additionally, researchers and developers can use various static and dynamic analysis tools to detect and fix front-end runtime errors, helping to enhance the security of the contracts.

3.1.6. Time Dependency

Time dependency attacks are a type of assault targeting smart contracts by exploiting time-related operations within the contract and the inherent characteristics of blockchain. In a blockchain, timestamps usually depend on the block generation time. However, mining nodes possess a certain degree of timestamp manipulation capability. Attackers may manipulate block timestamps to influence time-dependent smart-contract logic, such as timers, auction end times, or voting deadlines. This can result in contract behavior failure or outcomes that deviate from expectations, causing losses to contract participants.

In Figure 6, we name a deposit and withdrawal smart contract "Bank." Users can deposit funds by calling the *deposit*() function and withdraw funds using the *withdraw*() function. The *getBalance*() function can be utilized to query user balances. However, the contract is vulnerable to time dependency attacks. If an attacker calls the *deposit*() function before a user invokes the *withdraw*() function, they can disrupt the target contract's execution order and extract the deposit before the user's withdrawal. In the corresponding attack contract, the attacker creates a contract called "Attacker" and passes the target contract Bank's address in the constructor. Within the *attack*() function, the attacker first calls the *deposit*()function to deposit funds. Then, the attacker calls the *withdraw*()function to extract 1 ether of funds. Since the deposit has been committed and included in a new block but not yet added to the blockchain, the attacker successfully extracts the deposit before the withdrawal, causing losses to the deposit and withdrawal system users.

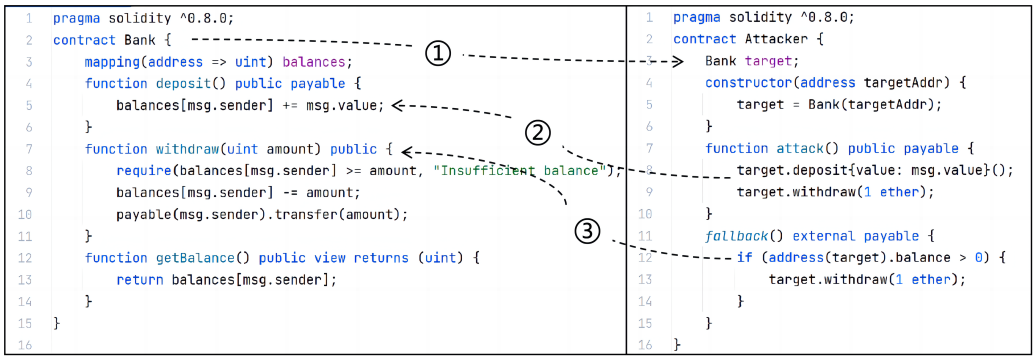

Figure 6. Front-end Runtime Error Vulnerability.

To prevent such attacks, developers should carefully consider the contract's time dependencies and adopt appropriate security measures during design and implementation. For instance, contracts can perform necessary state checks and input validation before transaction execution to prevent unnecessary interference or attacks. Moreover, users should be cautious about protecting their funds and information security when using contracts.

In summary, time-dependency attacks can pose a significant risk to smart contracts that rely on time-based logic. Developers need to be vigilant in addressing these vulnerabilities and implementing robust security measures when designing and building smart contracts. By taking the necessary precautions, both developers and users can help ensure the safety and integrity of smart contracts on the blockchain.

3.1.7. Other Vulnerabilities

Building upon the previously discussed smart-contract vulnerabilities, there are other common vulnerability types in practice. Here are several representative smart-contract vulnerabilities:

Delegatecall Vulnerability: Delegatecall operation serves as a mechanism for cross-contract invocation in smart contracts; however, inappropriate usage may lead to security vulnerabilities. Attackers, by crafting well-designed parameters for invocation, exploit the execution context of the target contract to perform malicious actions, resulting in asset theft or contract logic disruption.

Randomness Challenge: Due to the deterministic nature of blockchain, generating reliable random numbers within smart contracts poses a significant challenge. Attackers may predict or manipulate the random number generation process, thereby affecting the contract execution outcome and leading to asset loss or an unfair competitive environment.

Short Address Attack: A short address attack occurs when an attacker leverages the padding with zeros characteristic during data transmission, providing incomplete address information to mislead the contract. Consequently, the attacker bypasses the validation mechanism and incorrectly transfers assets to an address under their control.

Gas Limit and Optimization Issues: The execution of smart contracts requires the consumption of Gas, with the Gas Limit serving as the budget ceiling for contract execution. When handling complex logic without optimizing the contract code, Gas resources may be depleted, preventing the contract from functioning correctly. Moreover, attackers can construct high Gas-consuming transactions to perform a denial of service, thereby compromising contract availability.

The aforementioned list only covers some of the potential vulnerability types in smart contracts. As blockchain technology evolves, new vulnerabilities may emerge. Therefore, maintaining a focus on smart-contract security research and best practices is crucial for ensuring contract safety.

3.2. Development of Smart-Contract Security Detection

With the rapid development and widespread application of blockchain technology, smart contracts, as a core component, play a crucial role in implementing essential functions. However, the security issues of smart contracts urgently need to be addressed, as vulnerabilities may lead to severe economic losses and a crisis of trust. Against this backdrop, smart-contract security detection has gradually become a critical area of research and practice. This section will review the development history of smart-contract security detection, from the initial methods and tools to the emerging innovative technologies, revealing their evolution trends and future challenges.

In 2016, Luu et al. [30] were the first to introduce the smart-contract vulnerability detection tool Oyente, a symbolic execution-based tool designed specifically to discover potential security vulnerabilities. Oyente can detect common vulnerabilities in smart contracts, such as reentrancy attacks, transaction order dependencies, and timestamp dependencies. This pioneering work laid the foundation for research in the Ethereum smart-contract security domain and had a profound impact on subsequent related studies.

Two years later, Tikhomirov et al. [31] proposed a static analysis-based smart-contract vulnerability detection tool called SmartCheck. This tool uses ANTLR (a powerful parser generator for building language tools) and custom Solidity syntax to generate XML parse trees as an intermediate representation. Then, vulnerability patterns are identified by running XPath queries on the Intermediate Representation (IR). That same year, numerous other research achievements made significant progress in the field of smart-contract security. Breidenbach et al. [32] introduced Securify, a symbolic execution-based security analysis tool aimed at assessing the safety of Ethereum smart contracts. Securify employed technologies such as Program Query Language (PQL) and Domain Specific Language (DSL) to achieve automated analysis of smart contracts. Additionally, the tool utilized dependency graph analysis, compliance checks, and violation pattern recognition meth-

ods to identify potential security vulnerabilities within smart contracts. Brent et al. [33] presented an Ethereum smart-contract security analysis framework called Vandal. Vandal transformed EVM bytecode into semantic logic relations and used the Soufflé language for declarative security analysis. The paper also introduced a new decompilation technique for incremental control flow reconstruction. Kalra et al. [34] proposed the ZEUS framework, which combined abstract interpretation and symbolic model-checking techniques to verify the correctness and fairness of smart contracts. By developing a Solidity-to-LLVM bytecode converter and using LLVM pass separation for transformation and verification checks, the contract security verification achieved low false-positive rates and high analysis efficiency. In this year, besides the previously mentioned static detection methods, many dynamic detection algorithms emerged, such as fuzz testing techniques. These techniques generated random input data and detected potential vulnerabilities during program execution, thus supplementing the inadequacies of static analysis methods in smart-contract security vulnerability detection. Jiang et al. [35] introduced ContractFuzzer, a fuzz-testing framework specifically designed for detecting security vulnerabilities in Ethereum smart contracts. The paper provided a detailed description of ContractFuzzer's design, including input generation and test oracle analysis strategies, and demonstrated its effectiveness in the high-precision detection of seven types of Ethereum smart-contract vulnerabilities through experimental research.

One year later, Feist et al. [36] introduced a static analysis framework called Slither, designed to provide comprehensive information about Ethereum smart contracts. This framework achieves its goal by converting Solidity smart contracts into a dedicated intermediate representation called SlithIR. Employing a Static Single Assignment (SSA) form and a simplified instruction set, SlithIR streamlines the analysis process while preserving semantic information that might be lost during the conversion of Solidity to bytecode. Slither utilizes common program analysis techniques such as data flow analysis and taint tracking to uncover potential vulnerabilities and opportunities for code optimization. Chang et al. [37] proposed a method for automatically identifying critical paths in smart contracts and ranking them by importance called sCompile. This method uses symbolic execution to explore possible execution paths in smart contracts and identify those involving monetary transactions. By identifying critical paths through symbolic execution and ranking them by importance, paths that might violate safety or correctness are prioritized for analysis. In that year, the first machine-learning-based smart-contract vulnerability detection model, SmartEmbed [38], emerged. This tool consists of two phases: a model training phase and a prediction phase. The training phase comprises four main steps: tokenization, syntax parsing, code embedding, and similarity computation. Tokenization breaks down code into individual tokens; syntax parsing analyzes the code structure to identify syntactic components; code embedding maps each token and syntactic element to a high-dimensional vector space; and similarity computation compares the vectors of different code snippets to determine their similarity. In the prediction phase, SmartEmbed detects clones by identifying similar smart contracts based on embeddings and can also detect vulnerabilities by comparing contracts from the existing Ethereum blockchain or any contract provided by a developer to a vulnerability database. This tool can efficiently verify whether a given smart contract contains known vulnerabilities without the need to manually define vulnerability patterns.

Following this, the number of smart-contract vulnerability detection tools increased rapidly. For example, Huang et al. [39] introduced a vulnerability detection tool that combined graph embedding with bytecode. They normalized data and instructions, used simulated bytecode execution to track data flow and control flow, enforced contract slicing, and designed an unsupervised graph embedding algorithm to encode code graphs as comparable vectors, identifying potentially vulnerable smart contracts. Chen et al. [40] proposed DefectChecker, a method, and tool based on symbolic execution for detecting eight types of contract defects that could lead to undesirable behavior in Ethereum smart contracts, and validated its performance on open-source datasets. Chen et al. [41] auto-

matically recovered function signatures by utilizing the way EVM processes functions, identified parameter types using Type-Aware Symbolic Execution (TASE), and developed a tool called SigRec to recover function signatures from contract bytecode. Hu et al. [42] introduced a static defect detection method based on the Solidity language knowledge graph, called SoliDetector, which constructed an ontology layer and an instance layer, introduced defect patterns, designed inference rules, and used SPARQL queries to locate defects.

In recent years, machine-learning-based smart-contract vulnerability detection methods have attracted widespread attention and research. These methods take full advantage of machine-learning techniques, offering more efficient and accurate solutions for smart-contract security analysis. However, as Chapter 5 of this paper will specifically explore machine-learning-based smart-contract vulnerability detection techniques in-depth, further discussion will not be provided in this background section. In subsequent chapters, we will elaborate on the specific implementation of these methods and their application in the security analysis of smart contracts.

4. Machine-Learning Techniques

Machine learning is committed to enabling computers to accumulate new experience and knowledge by mining potential patterns in data, thereby improving their intelligence and enabling them to make decisions like humans. The application of machine-learning algorithms has become increasingly crucial in the field of smart-contract vulnerability detection. As various industries experience sustained growth in data demand and an escalating need for efficient data processing and analysis, numerous tailored machine-learning algorithms have emerged. These algorithms primarily rely on mathematical and statistical approaches to address optimization problems.

4.1. The Development of Machine Learning

The development of machine learning is shown in Figure 7. In 1943, McCulloch et al. [43] introduced a mathematical model that depicted the fundamental structure of artificial neural networks. This research laid the groundwork for the development of the neural network field and had a profound impact on later machine-learning techniques. In 1950, Turing proposed the "Turing Test" [44], marking the beginning of artificial intelligence as an important research area. In 1957, Rosenblatt et al. [45] introduced the perceptron, which initiated the study of computer neural networks. Hubel et al. [46] discovered a neural network structure that provided deep insights into understanding biological visual systems, significantly influencing later computer vision and neural network models.

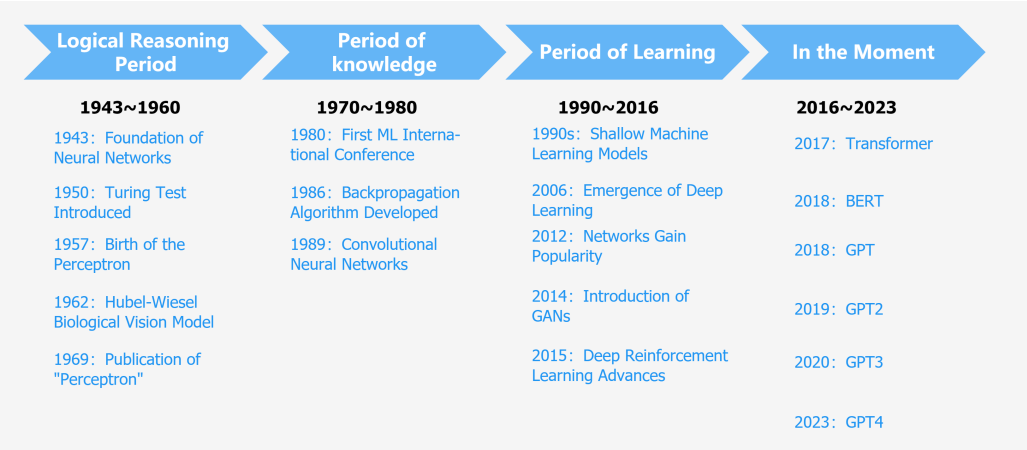

Figure 7. A timeline of the evolution of machine learning.

The first International Conference on Machine Learning in 1980 signified the global rise of the field. In 1986, Rumelhartd et al. [47] introduced a method to train multilayer neural networks using the backpropagation algorithm, resulting in a major breakthrough

in the neural network research field. The introduction of convolutional neural networks (CNN) in 1989 further propelled the field's development. In the 1990s, shallow machine-learning models such as logistic regression and support vector machines emerged. In 2006, the advent of deep learning models [48] revolutionized the machine-learning field. Deep learning models, by extracting high-level features from data through multilayer neural networks, significantly enhanced data representation capabilities and model accuracy. With the widespread application of deep learning techniques in computer vision, natural language processing, and other fields, it became a crucial driving force for the development of artificial intelligence. The 2012 Image Net competition breakthrough pushed deep learning to new heights.

In 2012, Graves et al. [49] first proposed the LSTM model, which addressed the vanishing and exploding gradient problems in recurrent neural networks (RNN) when processing long sequences, profoundly impacting sequence prediction and natural language processing fields. In 2014, Goodfellow et al. [50] introduced the concept of GANs, which included two competing neural networks: a generator network and a discriminator network. This framework brought innovation to generative model research. In 2015, Mnih et al. [51] proposed an algorithm called Deep Q-Network (DQN), which combined convolutional neural networks (CNN) with the Q-learning algorithm to process raw pixel inputs and action-value functions. This approach demonstrated the immense potential of combining deep learning with reinforcement learning for the first time.

In 2017, Vaswani et al. [52] introduced a novel neural network architecture based on the self-attention mechanism: Transformer. This had a profound impact on the Natural Language Processing (NLP) field. In 2018, the release of the BERT model [53] brought revolutionary changes to the natural language processing field. In the same year, OpenAI released GPT-1 and subsequently launched GPT-2 in 2019, GPT-3 in 2020, and GPT-4 in 2023, propelling natural language processing technology to unprecedented heights.

4.2. Machine-Learning Algorithms

We divide machine-learning techniques into four categories: Supervised Learning, Semi-Supervised Learning, Unsupervised Learning, and Reinforcement Learning. The general overview is shown in Figure 8. In what follows, we will discuss these four categories and their related methods in detail.

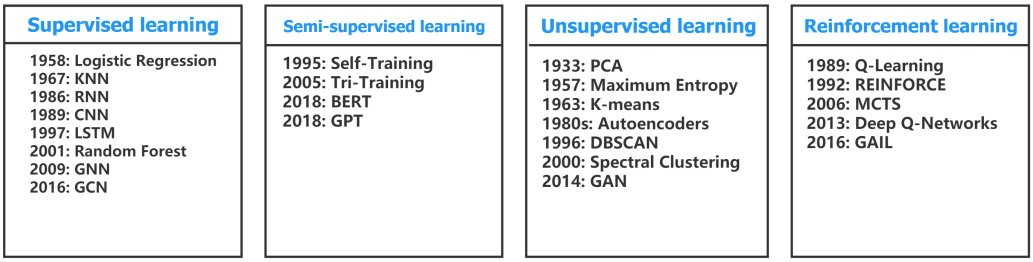

Figure 8. Classification overview of machine-learning methods.

4.2.1. Supervised Learning

In the domain of smart-contract security analysis, supervised learning approaches have achieved significant success and have been extensively employed in practical scenarios. This can be primarily attributed to the powerful capabilities of supervised learning algorithms in pattern recognition and knowledge representation, as well as their effective utilization of large volumes of labeled data. By training models to identify potential security vulnerabilities, supervised learning offers robust support for the security auditing of smart contracts.

In the following sections, we will provide a comprehensive understanding and inspiration for readers by elaborating on some representative supervised learning approaches employed in smart-contract security analysis.

Linear regression: This algorithm is a linear method used for modeling the relationship between a dependent variable and one or more independent variables, typically employed for predicting numerical outcomes. The fundamental equation is $y = kx + b$, where k represents the slope and b denotes the intercept. Linear regression offers simplicity, interpretability, and computational efficiency [54], and prediction problems constitute a classic application scenario of the linear regression algorithm [55].

Logistic regression: This algorithm is a linear method introduced by David et al. [56] in 1958 for modeling discrete target variables. The basic form of the model is $y = sigmoid(\beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k)$, where the sigmoid function is defined as $sigmoid(x) = \frac{1}{1+\exp(-x)}$, making logistic regression an effective method for binary classification tasks. Logistic regression provides ease of interpretability and can be easily extended to handle multi-class classification problems, currently being extensively applied in areas such as credit assessment and tumor diagnosis [57].

Support Vector Machines: This algorithm constitute a set of classical supervised learning methods, initially proposed by Corinna et al. [58], aimed at identifying the optimal hyperplane that maximizes the margin between distinct classes. SVM is formalized as a convex optimization problem, which can be expressed as $\min\|w\|^2$ subject to $y_i(w^T x_i + b) \geq 1$. These techniques are highly effective for classification and regression tasks in high-dimensional spaces and exhibit strong robustness against overfitting; thus, they are frequently applied in gene classification within the field of bioinformatics [59].

Random Forest: This algorithm is an ensemble learning method introduced by Breiman et al. [60] in 2001. It improves prediction accuracy by constructing multiple decision trees and combining their predictive outcomes using voting or averaging strategies. This approach has achieved success in numerous application domains, particularly being widely employed in classification and regression tasks [61].

K-Nearest Neighbors: This algorithm is an instance-based learning method proposed by Cover and Hart in 1967 [62]. It operates by computing the distances between a test data point and known data points using distance metrics, such as the Euclidean distance, identifying the nearest K neighbors, and classifying the test point based on the labels of these neighbors. This method holds classical significance in the field of pattern recognition [63].

Convolutional Neural Networks: This algorithm is a deep learning architecture specifically designed for processing grid-like data, such as images, introduced by LeCun et al. in 1989 [64]. CNNs employ convolutional layers to learn local features, pooling layers to reduce spatial dimensions, and fully connected layers for classification or regression [65]. The convolution operation is defined as $(f * g)(t) = \int f(\tau)g(t-\tau)d\tau$, enabling the network to effectively capture local patterns and hierarchical features. A classic application of CNNs is LeNet-5, which achieved breakthrough results in handwritten digit recognition tasks [66].

Graph Neural Networks: This algorithm is a deep learning method for processing graph data, proposed by Scarselli et al. in 2009 [67]. They learn node representations by performing information passing on nodes and edges. The core idea of GNNs is to multiply the adjacency matrix A of graph-structured data with the node feature matrix X, as in $A * X$. This approach finds broad applications in areas such as social network analysis, recommendation systems, and knowledge graphs [68].

Graph Convolutional Networks: This algorithm was introduced by Kipf et al. in 2016 [69] as a method of extending convolutional operations to graph-structured data. GCNs perform graph convolution operations using the adjacency matrix A and the node feature matrix X, as in $Z = \mathrm{ReLU}(A_{\mathrm{hat}} * X * W)$, where A_{hat} is the normalized adjacency matrix, X represents the node feature matrix, W is the weight matrix, and ReLU is the activation function. This method holds classical significance in tasks such as node classification, graph embedding, and link prediction [70].

Recurrent Neural Networks: This algorithm is a neural network method for processing sequence data, proposed by Rumelhart et al. in 1986 [71], capable of capturing temporal dependencies. The core idea of RNNs is to introduce recurrent connections in

the network's hidden layer, allowing information to be passed between time steps, as in $h_t = f(W * x_t + U * h_{(t-1)})$, where h_t represents the hidden state at time t, x_t represents the input at time t, W and U are weight matrices, and f is the activation function. RNNs find widespread applications in fields such as natural language processing, speech recognition, and time series prediction [72].

Long Short-Term Memory: This algorithm is a special type of Recurrent Neural Network (RNN) introduced by Hochreiter and Schmidhuber [73] in 1997 to address the vanishing gradient problem in long sequences. LSTMs control the storage and flow of information in cell states by introducing forget, input, and output gates, as in $f_t = \sigma(W_f * [h_{(t-1)}, x_t] + b_f)$. LSTMs are widely applied in tasks such as natural language processing, speech recognition, and time series prediction [74].

Gated Recurrent Units: This algorithm was proposed by Cho et al. in 2014 [75]. GRUs are a variant of LSTMs that reduce computational complexity by decreasing the number of gates while maintaining similar performance. GRUs control the storage and flow of information in cell states by introducing update and reset gates, as in $z_t = \sigma(W_z * [h_{(t-1)}, x_t] + b_z)$. GRUs exhibit good performance in tasks such as natural language processing, speech recognition, and time series prediction. Compared to Long Short-Term Memory (LSTM) networks, GRUs have fewer gates, thus reducing computational requirements while maintaining similar performance levels in many applications.

4.2.2. Semi-Supervised Learning

Although supervised learning methods have achieved significant results in smart-contract security detection, semi-supervised learning methods have been relatively less explored in this field. Semi-supervised learning methods combine labeled and unlabeled data during the training process, aiming to overcome the dependency on large amounts of labeled data in supervised learning. However, the application of semi-supervised learning methods in smart-contract security detection is limited by several factors. First, the complexity and diversity of smart-contract vulnerabilities may lead to insufficient generalization capabilities in semi-supervised learning methods. Second, the data distribution in this field may exhibit significant imbalances, which may negatively impact the performance of semi-supervised learning algorithms. We believe that although the application of semi-supervised learning methods in smart-contract security detection is relatively limited, they have potential advantages in dealing with data scarcity and reducing annotation costs. Therefore, in the future, semi-supervised learning methods may still play a role in smart-contract vulnerability detection, providing new solutions for security audits.

Self-training: This algorithm was first introduced by Yarowsky et al. [76]. Self-training methods initially train a base model on a small labeled dataset, then use the model to predict labels for unlabeled data. The main application areas of self-training methods include image classification and natural language processing tasks [77].

Tri-training: This algorithm proposed by Zhou et al. in 2005 [78], trains three models on a labeled dataset and uses each model to label unlabeled data. If two models agree on the label for a data point, the label is added to the labeled dataset, and the third model is retrained on the extended dataset. This process is repeated, improving model performance. Classic application areas of tri-training include text classification and named entity recognition [79].

BERT: Bidirectional Encoder Representations from Transformers, this algorithm was introduced by Devlin et al. in 2018 [80]. It is a pre-trained natural language processing model based on the Transformer architecture that undergoes pre-training on a large amount of unlabeled text using masked language modeling (MLM) and next sentence prediction (NSP) tasks. The key lies in its bidirectional context encoding, consisting of $E = (e_1, e_2, \ldots, e_n)$. It has been widely applied in areas such as text classification, named entity recognition, and question-answering systems [81].

GPT: Generative Pre-trained Transformer, this algorithm was proposed by Radford et al. in 2018 [82]. This is another pre-trained natural language processing model based on

the Transformer architecture that uses only unidirectional autoregressive language modeling tasks for pre-training. GPT models using the formula: $P(x) = \prod_i P(x_i \mid x_1, \ldots, x_{i-1})$. This formula denotes that, given the previous word sequence $x_1, \ldots, x_{i-1}$, the goal of GPT is to maximize the conditional probability of predicting the next word x_i [83]. The main application areas of GPT include text generation, text summarization, and machine translation [84].

4.2.3. Unsupervised Learning

Unsupervised learning is a type of machine-learning technique in which algorithms learn from and identify patterns in unlabelled data. In contrast to supervised learning, which relies on a dataset with labeled examples, unsupervised learning algorithms analyze the underlying structure or distribution of the data without any prior knowledge of the correct outputs.

K-means: This algorithm, proposed by MacQueen et al. [85], is a prototype-based iterative clustering method aimed at minimizing the sum of squared distances between data points within each cluster and their respective cluster centers. This method can be expressed by the following formula: $\text{argmin} \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$. Here, K represents the number of clusters, C_i denotes the i-th cluster, x signifies a data point, μ_i stands for the cluster center of the i-th cluster, and $|| \cdot ||$ represents the Euclidean distance. The K-means method has extensive applications in the field of market segmentation [86,87].

Spectral Clustering: This algorithm is a graph-theoretic clustering method that captures the complex structure of data in low-dimensional space by performing dimensionality reduction on the eigenvectors of the data's Laplacian matrix [88]. Spectral clustering has classical significance in the field of image segmentation [89].

Density-Based Spatial Clustering of Applications with Noise (DBSCAN): This algorithm, proposed by Ester et al. [90], is a density-based clustering algorithm. This method discovers clusters of arbitrary shapes without the need to specify the number of clusters by connecting dense regions and distinguishing noise points. The implementation steps of this method are: 1. Calculate the number of points within the ϵ-neighborhood of each data point. 2. Identify core points with an ϵ-neighborhood containing at least MinPts points. 3. Classify density-reachable core points as clusters, non-core points as the nearest cluster members, or noise.

Principal Component Analysis (PCA): This algorithm, proposed by Pearson et al. [91], is a linear dimensionality reduction technique that projects the original data onto a new orthogonal coordinate system, maximizing the data variance, thereby reducing the data dimensionality while preserving as much information as possible. PCA has classical significance in the field of face recognition [92].

Maximum Entropy: This algorithm, proposed by Jaynes et al. [93], is an information-theoretic statistical modeling technique that selects the most universal and least biased probability distribution by maximizing entropy under given constraints. The Maximum Entropy method has extensive applications in the part-of-speech tagging task [94].

Autoencoders: This algorithm, formally proposed by Rumelhart et al. [47], is an unsupervised neural network model that learns to compress and reconstruct data between the encoder and decoder, thereby achieving dimensionality reduction and feature extraction of data representation. The implementation process of autoencoders consists of three steps: encoding, decoding, and optimization. This method has extensive applications in the field of image denoising [95].

Generative Adversarial Network (GAN): This algorithm, proposed by Goodfellow et al. [50], consists of generative models based on adversarial training that learn to generate data similar to the true data distribution through a competitive process of simultaneously optimizing the generator and discriminator. A GAN has extensive applications in the field of image generation [96].

4.2.4. Reinforcement Learning

In the research of smart-contract security detection, reinforcement learning, as an important machine-learning method, has achieved significant results in multiple application fields in recent years. By interacting with the environment, reinforcement learning enables intelligent agents to autonomously explore the optimal strategy to maximize cumulative rewards in the long term. This section will briefly introduce some methods of reinforcement learning.

Q-Learning: This algorithm was introduced by Watkins et al. [97], as a model-free reinforcement learning algorithm that iteratively updates the action-value function $Q(s, a)$ to estimate the expected total return of executing a particular action in a given state, enabling the agent to select the optimal action based on Q-values. The general formula for Q-Learning is: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$. Here, α is the learning rate, and γ is the discount factor. Q-learning has classical significance in applications such as automatic control and network transmission [98,99].

Deep Q-Network: This algorithm was proposed by Mnih et al. [51]. It is a reinforcement learning algorithm that combines deep neural networks with Q-Learning, using neural networks to approximate the action-value function $Q(s, a)$ and handling high-dimensional, complex input state spaces, such as raw images. The general update formula for DQN is: $Q(s, a; \theta) \leftarrow Q(s, a; \theta) + \alpha[r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta)]$. Here, θ and θ' represent the parameters of the current and target neural networks, respectively.

REINFORCE: This algorithm was introduced by Williams et al. [100]. It is a reinforcement learning method that directly optimizes policy parameters by sampling trajectories to obtain unbiased estimates of the policy gradient and updating policy parameters using gradient ascent. Its core formula is: $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t$. Here, θ represents policy parameters, α is the learning rate, $\pi_\theta(a_t \mid s_t)$ is the probability of selecting action a_t in state s_t, and G_t is the cumulative reward starting from time step t. REINFORCE has classical significance in applications such as sequence generation and natural language processing [101].

MCTS: Monte Carlo Tree Search, this algorithm was proposed by Coulom et al. [102] as a search method based on Monte Carlo simulations that constructs a search tree and gradually finds an approximate optimal solution by balancing exploration and exploitation. MCTS does not have a general formula but primarily consists of four stages: Selection, Expansion, Simulation, and Backpropagation. AlphaGo is a prominent example of combining MCTS with deep learning [103]. In 2016, it defeated the world Go champion, Lee Sedol.

GAIL: Generative Adversarial Imitation Learning, this algorithm was introduced by Ho et al. [104]. It is a method that combines Generative Adversarial Networks (GANs) and reinforcement learning by training agents to acquire efficient policies through imitation learning of expert policies. The general formula for GAIL mainly includes generator (policy) loss and discriminator loss.

4.3. Comparing Different Kinds of Machine Learning

As shown in Table 1, we compared the advantages and disadvantages of the four types of supervised learning, semi-supervised learning, unsupervised learning, and reinforcement learning. See below for a detailed analysis.

Supervised Learning: This method is known for its high predictive accuracy and wide applicability across different domains. However, it requires a large labeled dataset to train effectively, which can be a limiting factor for some applications.

Semi-Supervised Learning: This approach benefits from high data utilization and reduced labeling costs compared to supervised learning, as it can work with partially labeled data. Nonetheless, it faces challenges such as increased algorithm complexity and dependence on underlying assumptions, which may affect its performance.

Table 1. Advantages and disadvantages of four machine-learning methods.

Machine-Learning Method	Advantages	Disadvantages
Supervised Learning	High predictive accuracy Wide applicability	Requires large labeled dataset
Semi-supervised Learning	High data utilization Reduced labeling cost	Algorithm complexity Dependence on assumptions
Unsupervised Learning	No need for labeled data Discovering latent structures	Limited predictive Challenging evaluation
Reinforcement Learning	Decision-making ability Adaptive learning	Computational complexity Slow convergence

Unsupervised Learning: The main advantages of this method are that it does not require labeled data and can discover latent structures within the data. However, its predictive performance is generally limited compared to supervised methods, and evaluating the quality of unsupervised models can be challenging.

Reinforcement Learning: This technique is particularly useful for decision-making and adaptive learning in dynamic environments. However, it comes with downsides, such as computational complexity and slow convergence, which may hinder its practicality in some situations.

5. Case Studies of ML-Based Smart-Contract Security Detection

In this section, we will explore the application of machine-learning techniques in the domain of smart-contract security detection. Machine-learning methods have demonstrated their powerful capabilities in various fields, consequently offering significant potential for smart-contract security detection. By reviewing relevant literature, we will delve into the application cases, advantages, and limitations of these methods in smart-contract security detection, ultimately providing valuable insights for future research and practice.

5.1. Document Retrieval

We conducted a comprehensive investigation into the application of machine learning in the domain of smart-contract security detection. In order to gain a thorough understanding of the latest advancements and trends in this field, we retrieved numerous relevant articles from authoritative databases such as Wiley, IEEE, Springer, ACM, and Elsevier. Given the limited number of such research works and their primary concentration within the last five years, we did not apply any time constraints during our search. While formulating the search strategy, we initially identified the main keywords and search terms, including "smart contract," "detection," "vulnerability," and "machine learning". To ensure the comprehensiveness of the search results, we expanded the keywords to encompass "bug," "fault," "security," "analysis," and more while also accounting for variations in tense and singular/plural forms by applying appropriate fuzziness. Based on these keywords and modifications, we constructed the following search formula:

("smart contract*") AND ("bug" OR "fault" OR "security" OR "vulnerability") AND ("detection" OR "analysis") AND ("machine learning" OR "deep learning" OR "reinforcement learning")

Employing this search formula with some adjustments, we retrieved a total of 176 articles. The number of related papers in each database over the years is shown in Figure 9. To ensure the quality and relevance of the selected literature, we adopted the following screening criteria: 1. Only include English literature; 2. Focus on empirical research pertaining to the topic; 3. Only include papers published in reputable academic journals or conferences. After the screening process, a total of 32 articles were ultimately included in this review.

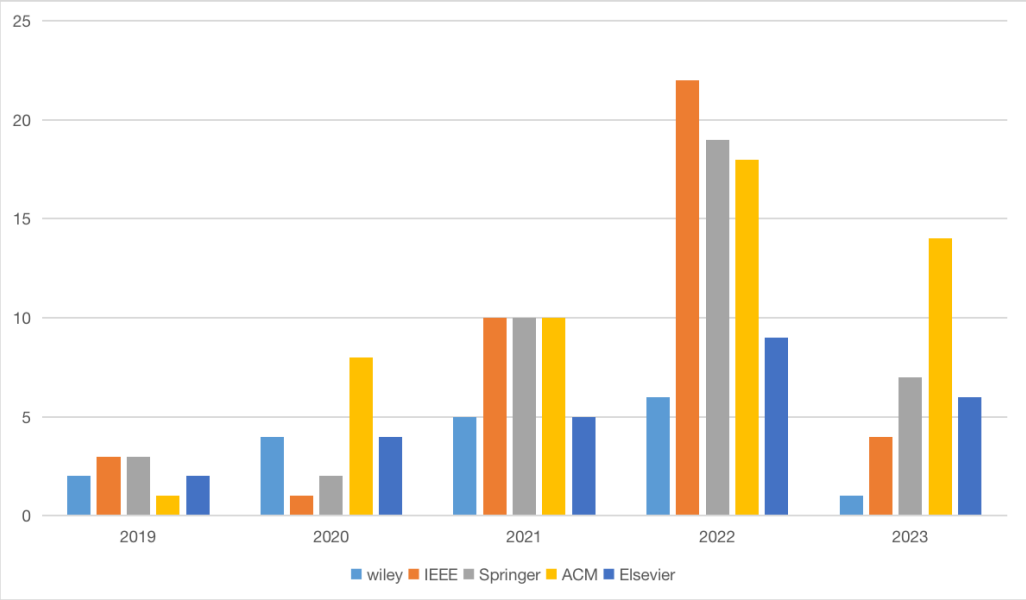

Figure 9. Related papers for each database.

5.2. Machine-Learning-Based Tools for Smart-Contract Vulnerability Detection

In this chapter, we will systematically present the smart-contract vulnerability detection frameworks that have successfully employed machine-learning techniques to date. Our discussion will be organized chronologically. Up to now, there have been 32 frameworks successfully utilizing machine-learning technologies for smart-contract vulnerability detection.

In 2019, Gao et al. [38] first introduced SmartEmbed, marking the inaugural research achievement in employing machine-learning techniques for detecting vulnerabilities in smart contracts. Their paper presented a web service tool based on code embedding and similarity detection methods. This tool achieved vulnerability detection by comparing the similarity between existing Solidity code on the Ethereum blockchain and the code embeddings of known vulnerabilities. Two years later, the authors refined their tool [105]. In their extended work, they further explored the application of machine-learning techniques in smart-contract security analysis and proposed several innovative improvements.

In 2020, Hao et al. introduced SCscan [106], a scanning system based on Support Vector Machines (SVM) for detecting vulnerabilities in blockchain smart contracts. The system aimed to identify potential security risks in smart contracts that could be exploited by attackers for illicit gains. In the same year, Lou et al. [107] proposed a Ponzi scheme detection method in smart contracts using an improved Convolutional Neural Network. They utilized a dataset of 3774 smart contracts for model training, including 132 Ponzi scheme contracts and 3642 legitimate smart contracts. Qian et al. [108] presented a deep learning approach based on bidirectional Long Short-Term Memory networks and attention mechanisms (BiLSTM-ATT) for the precise detection of reentrancy vulnerabilities. Furthermore, they introduced a contract fragment representation for smart contracts, which aids in capturing crucial semantic information and control flow dependencies.

In 2021, Hara et al. [109] employed machine-learning algorithms to detect Honeypots in Ethereum smart contracts. They proposed two feature extraction methods: one using TF-IDF to extract word features from the bytecode of the Ethereum blockchain and another using word2vec to extract distributed representations from the same data. These features could be used to detect Honeypots, and machine learning enhanced the detection performance. In the same year, Mi et al. [110] introduced a framework called VSCL for the automatic detection of vulnerabilities in smart contracts on blockchains. First, it leveraged a novel feature vector generation technique to extract information from the bytecode of smart contracts, as the source code of smart contracts is rarely available in public. Then,

the collected vectors were fed into their innovative deep neural network (DNN) based on metric learning to obtain detection results. Wang et al. [111] utilized deep learning techniques to automatically detect vulnerabilities in smart contracts. Their approach combined various code representations, such as code tokens, ASTs, and control flow graphs, and employed deep learning models for training and prediction. This method facilitated the learning of more comprehensive semantic information and enhanced the accuracy and completeness of vulnerability detection. Yu et al. [112] proposed a modular and systematic vulnerability detection framework based on deep learning named DeeSCVHunter. The framework focused on two types of smart-contract vulnerabilities: reentrancy and time dependence and introduced a novel concept called Vulnerability Candidate Slices (VCS) to help the model capture key points of vulnerabilities. Zhang et al. [113] presented a new classification model based on an improved CatBoost algorithm. The model employed a novel feature extraction pattern, delving deeper into the logic of smart-contract code. This approach could be used to detect Ponzi schemes during deployment and offer better performance, ultimately helping to prevent investor losses.

In 2022, Andrijasa et al. [114] employed deep reinforcement learning and multi-agent fuzz testing to develop improved techniques for detecting vulnerabilities in smart contracts. In the same year, Ashizawa et al. [115] introduced a machine-learning-based static analysis tool called Eth2Vec. This tool utilized neural networks to automatically learn features of vulnerable contracts and detect vulnerabilities in smart contracts by comparing the target contract code with the learned contract code. Gupta et al. [116] trained three different deep learning models, namely LSTM, ANN, and GRU, and applied them to predict the existence of vulnerabilities in smart contracts. These models were trained on known malicious and benign smart contracts, allowing for the automatic detection of vulnerabilities in new, unknown smart contracts. Hu et al. [117] proposed SCSGuard, a framework that employed machine-learning techniques to detect fraudulent behavior in smart contracts. SCSGuard leveraged the bytecode of smart contracts as a novel feature and utilized GRU networks and attention mechanisms to capture hidden information. Hwang et al. [118] introduced a new Convolutional Neural Network architecture, CodeNet, for smart-contract vulnerability detection. CodeNet addressed the issue of local information loss in existing CNN models by preserving the semantic and contextual information of smart contracts and demonstrated higher detection performance and faster detection times across various types of vulnerabilities. Li et al. [119] presented a new smart-contract vulnerability detection model called Link-DC. This model employed deep and cross networks to construct high-order nonlinear features and output these features to a fully connected layer to produce detection results. The model efficiently extracted features from raw data, thereby enhancing the performance and training efficiency of deep learning models. Liu et al. [120] proposed a heterogeneous graph transformation network for smart-contract anomaly detection (SHGTNs) to detect financial fraud on the Ethereum platform. They first extracted features to construct a Heterogeneous Information Network (HIN) of smart contracts, then fed the relation matrices obtained from learned meta-paths in the transformation network into a convolutional network, and finally utilized node embeddings for classification tasks. Nguyen et al. [121] proposed a novel heterogeneous graph representation approach called MANDO for learning the structure of heterogeneous contract graphs. MANDO developed a multiplex-path heterogeneous graph attention network to learn multi-layer embeddings of different types of nodes and their multiplex paths within the heterogeneous contract graph. This study extensively evaluated MANDO on a large-scale smart-contract dataset, showing that it improved the vulnerability detection results at the coarse-grained contract level compared to other techniques. Shakya et al. [122] introduced a vulnerability detection model named SmartMixModel, which employs machine-learning algorithms to detect vulnerabilities in Solidity smart contracts. This model extracts features at two levels—high-level syntactic features and low-level bytecode features of the smart contracts—to achieve more precise vulnerability detection. Wang et al. [123] proposed a machine-learning model called GVD-net to detect security vulnerabilities in Ethereum smart contracts. This model

takes the compiled smart-contract bytecode as input and uses a graph embedding-based machine-learning method for classification. Wu et al. [124] presented a deep learning-based framework for detecting vulnerabilities in Ethereum smart contracts, employing four deep learning models—CNN, LSTM, CNN-BiLSTM, and ResNets—to classify the source code of the smart contracts. Xu et al. [125] constructed a vulnerability detection model that utilized neural networks in machine learning, specifically bidirectional long short-term memory networks (BiLSTM), and introduced a hierarchical attention mechanism. This model takes code segments and account information of smart contracts as input, divides the input samples into three levels—word level, sentence level, and document level—and introduces attention mechanisms at different levels. By training the model to detect re-entry vulnerabilities in smart contracts, detection accuracy is improved and false positives are reduced. Zhang et al. [126] applied convolutional neural networks (CNN) to detect vulnerabilities in smart contracts. The authors transformed smart-contract vulnerabilities into image classification problems and converted bytecodes into numerical images based on predefined rules. They then trained and classified these numerical images using CNN to detect vulnerabilities in smart contracts. Zheng et al. [127] built a larger dataset and extracted numerous independent features from multiple perspectives, including bytecode, semantics, and developers, that were not related to transactions. They then constructed a multi-view cascading ensemble model (MulCas) using machine-learning methods, enabling their model to identify Ponzi schemes at the time of smart-contract creation. Zhou et al. [128] first investigated the classification of security issues related to smart contracts in BIoT scenarios. To address these security issues and overcome the limitations of existing methods, they proposed a tree-based machine-learning vulnerability detection (TMLVD) approach to perform vulnerability analysis of smart contracts. TMLVD inputs an intermediate representation derived from the abstract syntax tree (AST) of smart contracts into a tree-based training network to build a prediction model. This model captures multi-dimensional features to identify vulnerable smart contracts. The detection phase can be quickly implemented with limited computational resources while ensuring the accuracy of the detection results. The experimental evaluation demonstrated the effectiveness and efficiency of TMLVD on a dataset composed of Ethereum smart contracts.

In 2023, Cai et al. [129] proposed a graph neural network (GNN)-based method for smart-contract vulnerability detection. First, by combining abstract syntax trees (AST), control flow graphs (CFG), and program dependency graphs (PDG), they built a graph representation for smart-contract functions that included both syntactic and semantic features. To further enhance the representational power of their method, they performed program slicing to normalize the graph and eliminate redundancy unrelated to vulnerabilities. They then employed a bidirectional gated graph neural network model with mixed attention pooling to identify potential vulnerabilities in smart-contract functions. Jiang et al. [130] introduced VDDL, which utilized a multi-layer bidirectional Transformer structure as its model framework, involving multi-head attention and masking mechanisms. Multi-head attention was applied in the encoder and decoder layers. The masking mechanism enabled deep bidirectional training representations by randomly masking input tokens and predicting masked tokens using context. Furthermore, VDDL incorporated CodeBERT, a large-scale dual-modal pretraining model for natural language and programming language, to enhance training results. Jie et al. [131] used a multi-modal artificial intelligence framework to detect vulnerabilities in smart contracts. The framework combined various techniques such as natural language processing, image processing, and code analysis and employed machine-learning algorithms like support vector machines (SVM) and long short-term memory networks (LSTM) to improve vulnerability detection accuracy and efficiency. Liu et al. [132] explored the use of graph neural networks and expert knowledge for detecting smart-contract vulnerabilities. Specifically, they transformed the rich control and data flow semantics of the source code into contract graphs. To highlight key nodes in the graph, they further designed a node elimination phase to normalize the graph. They then proposed a novel temporal message propagation network to extract graph features from

the normalized graph, combining these features with designed expert patterns to produce the final detection system. Extensive experiments were performed on all smart contracts with source code on the Ethereum and VNT Chain platforms. Su et al. [133] proposed a reinforcement learning-based vulnerability-guided fuzz testing approach called RLF, used for generating vulnerability transaction sequences to detect complex vulnerabilities in smart contracts. Specifically, they first modeled the process of fuzz testing smart contracts as a Markov decision process, constructing a reinforcement learning framework. They then designed a reward that considered vulnerabilities and code coverage to effectively guide the fuzzer in generating specific transaction sequences to reveal vulnerabilities, especially those related to multiple functions. Sun et al. [134] introduced a novel smart-contract vulnerability detection framework called ASSBert, which combined active learning and semi-supervised learning to address the issue of insufficient labeled data. ASSBert employed bidirectional encoder representations from Transformers (BERT). Active learning was responsible for filtering highly uncertain code data from unlabeled sol files and manually annotating them, while semi-supervised learning continuously selected a certain number of high-confidence unlabeled code data from unlabeled sol files and included them in the training set after pseudo-labeling. Zhang et al. [135] proposed a deep learning-based two-stage smart-contract debugger, ReVulDL, for detecting and locating re-entry vulnerabilities. ReVulDL integrated vulnerability detection and location into a unified debugging process. The detection stage leveraged a graph-based pre-trained model to learn complex relationships in the propagation chain; the location stage applied interpretable machine-learning to pinpoint suspicious statements.

5.3. Comparative Analysis of Existing Machine-Learning-Based Smart-Contract Vulnerability Detection Tools

As shown in Table 2, this is a comparative analysis table about machine learning-based smart contract vulnerability detection tools. Within the table, we can observe detailed comparisons of various research studies or tools proposed by different authors and teams. The table encompasses several key attributes: first, references are provided for readers to easily locate relevant materials; second, the machine-learning methods employed are specified; subsequently, the size of the datasets is included, which aids in understanding the impact of data scale on model performance and reliability; finally, the machine-learning classification approaches utilized by the tools, such as supervised learning, semi-supervised learning, or reinforcement learning, are presented, assisting in comprehending the distinctions between different methods as well as their strengths and weaknesses in addressing smart-contract vulnerability detection issues. Through this table, we can gain a comprehensive understanding of various machine-learning-based smart-contract vulnerability detection tools and provide valuable references for further research and practice.

Table 2. Comparison of machine-learning-based smart-contract vulnerability detection tools.

Model	Method	Dataset Size	Classification	Year
Gao et al. [38]	Code Embedding	22,275	Supervised Learning	2019
Hao et al. [106]	SVM	Not provided	Supervised Learning	2020
Lou et al. [107]	CNN	3774	Supervised Learning	2020
Qian et al. [108]	Bi-LSTM	2000	Supervised Learning	2020
Hara et al. [109]	Machine Learning	151,935	Supervised Learning	2021
Mi et al. [110]	DNN	40	Supervised Learning	2021
Wang et al. [111]	Deep Learning	More than 2000	Supervised Learning	2021

Table 2. *Cont.*

Model	Method	Dataset Size	Classification	Year
Yu et al. [112]	Deep Learning	More than 50,932	Supervised Learning	2021
Zhang et al. [113]	CatBoost and Random Forrest	3774	Supervised Learning	2021
Andrijasa et al. [114]	Deep Reinforcement Learning	Not provided	Reinforcement Learning	2022
Ashizawa et al. [115]	Machine Learning	95,152	Supervised Learning	2022
Gupta et al. [116]	LSTM, ANN and GRU	7000	Supervised Learning	2022
Hu et al. [117]	GRU AND Attention	More than 1000	Supervised Learning	2022
Hwang et al. [118]	Machine Learning	47,518	Supervised Learning	2022
Li et al. [119]	Deep Learning	3000	Supervised Learning	2022
Liu et al. [120]	HGTNs	1382	Supervised Learning	2022
Nguyen et al. [121]	Machine Learning	47,891	Supervised Learning	2022
Shakya et al. [122]	Machine Learning	70,000	Supervised Learning	2022
Wang et al. [123]	Graph Embedding and Machine Learning	Not provided	Supervised Learning	2022
Wu et al. [124]	Deep Learning	11,881	Supervised Learning	2022
Xu et al. [125]	Bi-LSTM	36,232	Supervised Learning	2022
Zhang et al. [126]	CNN	Not provided	Supervised Learning	2022
Zheng et al. [127]	Machine Learning	10,349	Supervised Learning	2022
Zhou et al. [128]	Machine Learning and AST	20,567	Supervised Learning	2022
Cai et al. [129]	GNN	More than 9369	Supervised Learning	2023
Jiang et al. [130]	BERT	47,038	Semi-Supervised Learning	2023
Jie et al. [131]	Machine Learning	Not provided	Supervised Learning	2023
Liu et al. [132]	GNN and Expert Knowledge	40,932	Supervised Learning	2023
Su et al. [133]	Reinforcement Learning and Fuzzing	Not provided	Reinforcement Learning	2023
Sun et al. [134]	BERT	20,829	Semi-Supervised Learning	2023
Zhang et al. [135]	Deep Learning	47,398	Supervised Learning	2023

6. Discussion

In this section, we will comprehensively explore the three key research questions proposed in this paper, delving into existing methods and techniques for each question to provide valuable insights for researchers and practitioners.

RQ1: There are numerous methods that can be applied to the field of smart-contract security detection, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Most of the currently implemented machine-learning-based smart-contract security detection methods primarily rely on supervised learning, which is the most common machine-learning technique in smart-contract vulnerability detection. The main advantage of supervised learning methods is that they are trained on a large amount of labeled data, allowing the patterns and features learned from the training

data to be effectively applied to new data. However, the downside of these methods is that they require a substantial amount of labeled data for training. Semi-supervised learning approaches are less commonly used in smart-contract vulnerability detection but have great potential. The advantage of these methods is that the pre-training process does not require labeled training data, enabling the development of such methods based on pre-trained large models. The limitation of semi-supervised learning methods is that they may struggle to capture specific vulnerability features. Unsupervised learning methods are rarely applied in smart-contract vulnerability detection. The advantage of these methods is that they do not require labeled training data, while their limitation lies in their potential difficulty in capturing specific vulnerability features. Reinforcement learning is a machine-learning technique based on the interaction between an agent and its environment.

RQ2: Supervised learning, semi-supervised learning, and reinforcement learning methods have all been applied in the field of smart-contract security detection. However, the vast majority of applications are actually based on supervised learning methods. This paper posits that this is likely due to the current maturity of supervised learning methods and because labeled datasets enable achieving better results with smaller amounts of data. However, as the amount of data in smart-contract datasets continues to grow, semi-supervised learning is also becoming a promising research direction.

RQ3: In the field of smart-contract security detection, machine learning can be combined with static analysis, dynamic analysis, and fuzz testing methods. To integrate with static analysis, features (such as code patterns and function calls) can be extracted from the smart contract's source code or bytecode and used to train machine-learning models to identify potential vulnerabilities. To combine with dynamic analysis, runtime data (such as state changes and transaction flows) can be collected during contract execution and used in conjunction with machine-learning models to detect anomalous behavior and potential vulnerabilities. To integrate with fuzz testing methods, random or semi-random input data can be generated, and the contract execution results can be observed. Machine learning can then be employed to analyze the execution process and outcomes, automatically discovering new vulnerabilities or abnormal behaviors.

7. Conclusions and Future Work

Although machine learning has made some progress in the detection of defects in smart contracts, there is still a noticeable gap in the literature regarding a comprehensive review of machine-learning-based smart-contract defect detection. To address this shortcoming, this paper innovatively delves into the application of machine learning in the field of smart-contract security detection, aiming to provide valuable references and inspiration for researchers. The paper conducts an in-depth analysis and classification of machine-learning techniques, exploring the effectiveness of different technologies in smart-contract defect detection. Furthermore, this paper investigates and compares existing machine-learning-based smart-contract defect detection models. In future research, we will explore the possibility of combining machine-learning techniques with formal methods.

Author Contributions: Conceptualization, J.X.; methodology, Q.L.; formal analysis, K.G.; data curation, J.W.; writing—original draft preparation, F.J. and K.C.; writing—review and editing, F.J. and K.C.; supervision, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the Jiangxi Provincial Natural Science Foundation under Grant No. 20224ACB202007, and in part by the Jiangxi Province 03 Special Project and 5G Project under Grant No. 20224ABC03A13.

Acknowledgments: The authors would like to thank the editor and the reviewers for the valuable and constructive comments. They have been very helpful in the revision of this paper and allowed us to improve the technical content and presentation quality.

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. In *Decentralized Business Review*; Satoshi Nakamato Institute: Austin, TX, USA, 2008; p. 21260.
2. Gad, A.G.; Mosa, D.T.; Abualigah, L.; Abohany, A.A. Emerging Trends in Blockchain Technology and Applications: A Review and Outlook. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 6719–6742. [CrossRef]
3. Sahoo, S.; Kumar, A.; Mishra, R.; Tripathi, P. Strengthening Supply Chain Visibility With Blockchain: A PRISMA-Based Review. *IEEE Trans. Eng. Manag.* **2022**, 1–17. [CrossRef]
4. Liu, Y.; Qian, K.; Wang, K.; He, L. BCmaster: A Compatible Framework for Comprehensively Analyzing and Monitoring Blockchain Systems in IoT. *IEEE Internet Things J.* **2022**, *9*, 22529–22546. [CrossRef]
5. Tyagi, A.K.; Dananjayan, S.; Agarwal, D.; Thariq Ahmed, H.F. Blockchain—Internet of Things Applications: Opportunities and Challenges for Industry 4.0 and Society 5.0. *Sensors* **2023**, *23*, 947. [CrossRef]
6. Xu, S.; Guo, C.; Hu, R.Q.; Qian, Y. Blockchain-Inspired Secure Computation Offloading in a Vehicular Cloud Network. *IEEE Internet Things J.* **2022**, *9*, 14723–14740. [CrossRef]
7. Liu, D.; Alahmadi, A.; Ni, J.; Lin, X.; Shen, X. Anonymous Reputation System for IIoT-Enabled Retail Marketing Atop PoS Blockchain. *IEEE Trans. Ind. Inf.* **2019**, *15*, 3527–3537. [CrossRef]
8. Han, Y.; Zhang, Y.; Vermund, S.H. Blockchain Technology for Electronic Health Records. *Int. J. Environ. Res. Public Health* **2022**, *19*, 15577. [CrossRef]
9. Jaiman, V.; Urovi, V. A Consent Model for Blockchain-Based Health Data Sharing Platforms. *IEEE Access* **2020**, *8*, 143734–143745. [CrossRef]
10. Liu, X.; Wang, W.; Niyato, D.; Zhao, N.; Wang, P. Evolutionary Game for Mining Pool Selection in Blockchain Networks. *IEEE Wirel. Commun. Lett.* **2018**, *7*, 760–763. [CrossRef]
11. Gurzhii, A.; Islam, A.K.M.N.; Haque, A.K.M.B.; Marella, V. Blockchain Enabled Digital Transformation: A Systematic Literature Review. *IEEE Access* **2022**, *10*, 79584–79605. [CrossRef]
12. Sunny, F.A.; Hajek, P.; Munk, M.; Abedin, M.Z.; Satu, M.S.; Efat, M.I.A.; Islam, M.J. A systematic review of blockchain applications. *IEEE Access* **2022**, *10*, 59155–59177. [CrossRef]
13. Blockchain Boosts Global Economy: A PWC Digital Report. Available online: https://www.pwc.com/gx/en/news-room/press-releases/2020/blockchain-boost-global-economy-track-trace-trust.html (accessed on 13 October 2020).
14. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
15. Cheng, H.; Hu, Q.; Zhang, X.; Yu, Z.; Yang, Y.; Xiong, N. Trusted Resource Allocation Based on Smart Contracts for Blockchain-Enabled Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 7904–7915. [CrossRef]
16. Zhu, S.; Cai, Z.; Hu, H.; Li, Y.; Li, W. zkCrowd: A Hybrid Blockchain-Based Crowdsourcing Platform. *IEEE Trans. Ind. Inf.* **2020**, *16*, 4196–4205. [CrossRef]
17. Saini, A.; Zhu, Q.; Singh, N.; Xiang, Y.; Gao, L.; Zhang, Y. A Smart-Contract-Based Access Control Framework for Cloud Smart Healthcare System. *IEEE Internet Things J.* **2021**, *8*, 5914–5925. [CrossRef]
18. Atzei, N.; Bartoletti, M.; Cimoli, T. A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, 22–29 April 2017, Proceedings 6*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 164–186.
19. Chen, H.; Pendleton, M.; Njilla, L.; Xu, S. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Comput. Surv. CSUR* **2020**, *53*, 1–43. [CrossRef]
20. Liu, J.; Liu, Z. A survey on security verification of blockchain smart contracts. *IEEE Access* **2019**, *7*, 77894–77904. [CrossRef]
21. Kabla, A.H.H.; Anbar, M.; Manickam, S.; Alamiedy, T.A.; Cruspe, P.B.; Al-Ani, A.K.; Karupayah, S. Applicability of intrusion detection system on Ethereum attacks: A comprehensive review. *IEEE Access* **2022**, *10*, 71632–71655. [CrossRef]
22. Rameder, H.; Di Angelo, M.; Salzer, G. Review of automated vulnerability analysis of smart contracts on Ethereum. *Front. Blockchain* **2022**, *5*, 814977. [CrossRef]
23. Kushwaha, S.S.; Joshi, S.; Singh, D.; Kaur, M.; Lee, H.N. Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access* **2022**, *10*, 6605–6621. [CrossRef]
24. Krichen, M.; Lahami, M.; Al-Haija, Q.A. Formal Methods for the Verification of Smart Contracts: A Review. In Proceedings of the 15th International Conference on Security of Information and Networks (SIN), Sousse, Tunisia, 11–13 November 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 01–08.
25. Miller, A.; Cai, Z.; Jha, S. Smart contracts and opportunities for formal methods. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, 5–9 November 2018, Proceedings, Part IV 8*; Springer International Publishing: Cham, Switzerland, 2018; pp. 280–299.
26. Ahmed, S.J.; Taha, D.B. Machine Learning for Software Vulnerability Detection: A Survey. In Proceedings of the 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM), Mosul, Iraq, 31 August–1 September 2022; pp. 66–72.
27. Pan, Z.; Mishra, P. A survey on hardware vulnerability analysis using machine learning. *IEEE Access* **2022**, *10*, 49508–49527. [CrossRef]
28. Zeng, P.; Lin, G.; Pan, L.; Tai, Y.; Zhang, J. Software vulnerability analysis and discovery using deep learning techniques: A survey. *IEEE Access* **2020**, *8*, 197158–197172. [CrossRef]

29. Lin, G.; Wen, S.; Han, Q.L.; Zhang, J.; Xiang, Y. Software vulnerability detection using deep neural networks: A survey. *Proc. IEEE* **2020**, *108*, 1825–1848. [CrossRef]
30. Luu, L.; Chu, D.-H.; Olickel, H.; Saxena, P.; Hobor, A. Making Smart Contracts Smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 254–269.
31. Tikhomirov, S.; Voskresenskaya, E.; Ivanitskiy, I.; Takhaviev, R.; Marchenko, E.; Alexandrov, Y. SmartCheck: Static Analysis of Ethereum Smart Contracts. In Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, Gothenburg, Sweden, 27 May 2018; pp. 9–16.
32. Tsankov, P.; Dan, A.; Cohen, D.D.; Gervais, A.; Buenzli, F.; Vechev, M. Securify: Practical Security Analysis of Smart Contracts. *arXiv* **2018**, arXiv:1806.01143.
33. Brent, L.; Jurisevic, A.; Kong, M.; Liu, E.; Gauthier, F.; Gramoli, V.; Holz, R.; Scholz, B. Vandal: A Scalable Security Analysis Framework for Smart Contracts. *arXiv* **2018**, arXiv:1809.03981.
34. Kalra, S.; Goel, S.; Dhawan, M.; Sharma, S. ZEUS: Analyzing Safety of Smart Contracts. In Proceedings of the 2018 Network and Distributed System Security Symposium, San Diego, CA, USA, 18–21 February 2018.
35. Jiang, B.; Liu, Y.; Chan, W.K. ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection. In Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 259–269. [CrossRef]
36. Feist, J.; Grieco, G.; Groce, A. Slither: A Static Analysis Framework For Smart Contracts. In Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Montreal, QC, Canada, 26 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 8–15. [CrossRef]
37. Chang, J.; Gao, B.; Xiao, H.; Sun, J.; Cai, Y.; Yang, Z. sCompile: Critical Path Identification and Analysis for Smart Contracts. *arXiv* **2019**, arXiv:1808.00624.
38. Gao, Z.; Jayasundara, V.; Jiang, L.; Xia, X.; Lo, D.; Grundy, J. SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding. In Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 29 September–4 October 2019; pp. 394–397.
39. Huang, J.; Han, S.; You, W.; Shi, W.; Liang, B.; Wu, J.; Wu, Y. Hunting Vulnerable Smart Contracts via Graph Embedding Based Bytecode Matching. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 2144–2156. [CrossRef]
40. Chen, J.; Xia, X.; Lo, D.; Grundy, J.; Luo, X.; Chen, T. DefectChecker: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode. *IEEE Trans. Softw. Eng.* **2022**, *48*, 2189–2207. [CrossRef]
41. Chen, T.; Li, Z.; Luo, X.; Wang, X.; Wang, T.; He, Z.; Fang, K.; Zhang, Y.; Zhu, H.; Li, H.; et al. SigRec: Automatic Recovery of Function Signatures in Smart Contracts. *IEEE Trans. Softw. Eng.* **2022**, *48*, 3066–3086. [CrossRef]
42. Hu, T.; Li, B.; Pan, Z.; Qian, C. Detect Defects of Solidity Smart Contract Based on the Knowledge Graph. *IEEE Trans. Reliab.* **2023**, 1–17. [CrossRef]
43. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
44. Turing, A.M. *Computing Machinery and Intelligence*; Springer: Dordrecht, The Netherlands, 2009; pp. 23–65.
45. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408. [CrossRef]
46. Hubel, D.H.; Wiesel, T.N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* **1962**, *160*, 106–154. [CrossRef] [PubMed]
47. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
48. Hinton, G.E.; Salakhutdinov, R.R. Reducing the Dimensionality of Data with Neural Networks. *Science* **2006**, *313*, 504–507. [CrossRef] [PubMed]
49. Graves, A.; Graves, A. Long Short-Term Memory. In *Supervised Sequence Labelling with Recurrent Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 37–45.
50. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial networks. *Commun. ACM* **2020**, *63*, 139–144. [CrossRef]
51. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
52. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2017.
53. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805v2.
54. Kim, S.J.; Bae, S.J.; Jang, M.W. Linear Regression Machine Learning Algorithms for Estimating Reference Evapotranspiration Using Limited Climate Data. *Sustainability* **2022**, *14*, 11674. [CrossRef]
55. Maulud, D.; Abdulazeez, A.M. A review on linear regression comprehensive in machine learning. *J. Appl. Sci. Technol. Trends* **2020**, *1*, 140–147. [CrossRef]
56. Cox, D.R. The regression analysis of binary sequences. *J. R. Stat. Soc. Ser. B Methodol.* **1958**, *20*, 215–232. [CrossRef]

57. Aniche, M.; Maziero, E.; Durelli, R.; Durelli, V.H. The effectiveness of supervised machine learning algorithms in predicting software refactoring. *IEEE Trans. Softw. Eng.* **2020**, *48*, 1432–1450. [CrossRef]
58. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
59. Gigović, L.; Pourghasemi, H.R.; Drobnjak, S.; Bai, S. Testing a new ensemble model based on SVM and random forest in forest fire susceptibility assessment and its mapping in Serbia's Tara National Park. *Forests* **2019**, *10*, 408. [CrossRef]
60. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
61. González, C.; Astudillo, C.A.; López-Cortés, X.A.; Maldonado, S. Semi-supervised learning for MALDI–TOF mass spectrometry data classification: An application in the salmon industry. *Neural Comput. Appl.* **2023**, *35*, 1–11. [CrossRef]
62. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21–27. [CrossRef]
63. Gallego, A.J.; Rico-Juan, J.R.; Valero-Mas, J.J. Efficient k-nearest neighbor search based on clustering and adaptive k values. *Pattern Recognit.* **2022**, *122*, 108356. [CrossRef]
64. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
65. Yudistira, N.; Kavitha, M.S.; Kurita, T. Weakly-Supervised Action Localization, and Action Recognition Using Global–Local Attention of 3D CNN. *Int. J. Comput. Vis.* **2022**, *130*, 2349–2363. [CrossRef]
66. Wei, G.; Li, G.; Zhao, J.; He, A. Development of a LeNet-5 gas identification CNN structure for electronic noses. *Sensors* **2019**, *19*, 217. [CrossRef] [PubMed]
67. Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, Montreal, QC, Canada, 31 July–4 August 2005; IEEE: Piscataway, NJ, USA, 2015; Volume 2, pp. 729–734.
68. Ciano, G.; Rossi, A.; Bianchini, M.; Scarselli, F. On inductive–transductive learning with graph neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 758–769. [CrossRef] [PubMed]
69. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv* **2016**, arXiv:1609.02907.
70. Chen, T.; Zhang, X.; You, M.; Zheng, G.; Lambotharan, S. A GNN-based supervised learning framework for resource allocation in wireless IoT networks. *IEEE Internet Things J.* **2021**, *9*, 1712–1724. [CrossRef]
71. Elman, J.L. Finding structure in time. *Cogn. Sci.* **1990**, *14*, 179–211. [CrossRef]
72. Mikolov, T.; Karafiát, M.; Burget, L.; Cernocký, J.; Khudanpur, S. Recurrent neural network based language model. In Proceedings of the Interspeech, Makuhari, Japan, 12–16 September 2010; Volume 2, pp. 1045–1048.
73. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
74. Polat, H.; Türkoğlu, M.; Polat, O.; Şengür, A. A novel approach for accurate detection of the DDoS attacks in SDN-based SCADA systems based on deep recurrent neural networks. *Expert Syst. Appl.* **2022**, *197*, 116748. [CrossRef]
75. Cho, K.; Van Merriënboer, B.; Bahdanau, D.; Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv* **2014**, arXiv:1409.1259.
76. Yarowsky, D. Unsupervised word sense disambiguation rivaling supervised methods. In Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, Cambridge, MA, USA, 26–30 June 1995; pp. 189–196.
77. Xu, H.; Li, L.; Guo, P. Semi-supervised active learning algorithm for SVMs based on QBC and tri-training. *J. Ambient Intell. Humaniz. Comput.* **2021**, *12*, 8809–8822. [CrossRef]
78. Zhou, Z.H.; Li, M. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 1529–1541. [CrossRef]
79. Ning, X.; Wang, X.; Xu, S.; Cai, W.; Zhang, L.; Yu, L.; Li, W. A review of research on co-training. *Concurr. Comput. Pract. Exp.* **2021**, e6276. [CrossRef]
80. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
81. Yang, N.; Jo, J.; Jeon, M.; Kim, W.; Kang, J. Semantic and explainable research-related recommendation system based on semi-supervised methodology using BERT and LDA models. *Expert Syst. Appl.* **2022**, *190*, 116209. [CrossRef]
82. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. *OpenAI Technical Report*. 2018. Available online: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf (accessed on 15 March 2023).
83. Floridi, L.; Chiriatti, M. GPT-3: Its nature, scope, limits, and consequences. *Minds Mach.* **2020**, *30*, 681–694. [CrossRef]
84. Katz, D.M.; Bommarito, M.J.; Gao, S.; Arredondo, P. GPT-4 Passes the Bar Exam. SSRN. 2023. Available online: https://ssrn.com/abstract=4389233 (accessed on 15 March 2023).
85. MacQueen, J. Classification and analysis of multivariate observations. In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Los Angeles, CA, USA, 21–23 June 1967; University of California: Berkeley, CA, USA, 1967; Volume 1, pp. 281–297.
86. Punj, G.; Stewart, D.W. Cluster analysis in marketing research: Review and suggestions for application. *J. Mark. Res.* **1983**, *20*, 134–148. [CrossRef]
87. Dolnicar, S. A review of unquestioned standards in using cluster analysis for data-driven market segmentation. *J. Mark. Theory Pract.* **2002**, *10*, 1–12.
88. Von Luxburg, U. A tutorial on spectral clustering. *Stat. Comput.* **2007**, *17*, 395–416. [CrossRef]

89. Ng, A.; Jordan, M.; Weiss, Y. On spectral clustering: Analysis and an algorithm. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver BC, Canada, 3–8 December 2001; Volume 14, pp. 849–856.
90. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, Portland, OR, USA, 2–4 August 1996; Volume 96, pp. 226–231.
91. Pearson, K. On lines and planes of closest fit to systems of points in space. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **1901**, *2*, 559–572. [CrossRef]
92. Turk, M.; Pentland, A. Eigenfaces for recognition. *J. Cogn. Neurosci.* **1991**, *3*, 71–86. [CrossRef]
93. Jaynes, E.T. Information theory and statistical mechanics. *Phys. Rev.* **1957**, *106*, 620. [CrossRef]
94. Toutanova, K.; Klein, D.; Manning, C.D.; Singer, Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Edmonton, AB, Canada, 27 April–1 May 2003; pp. 252–259.
95. Masci, J.; Meier, U.; Cireşan, D.; Schmidhuber, J. Stacked convolutional auto-encoders for hierarchical feature extraction. In Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN 2011), Espoo, Finland, 14–17 June 2011; Part I 21; Springer: Berlin/Heidelberg, Germany, 2011; pp. 52–59.
96. Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *arXiv* **2017**, arXiv:1710.10196.
97. Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D. Thesis, University of Cambridge, Cambridge, UK, 2017.
98. Clifton, J.; Laber, E. Q-learning: Theory and applications. *Annu. Rev. Stat. Its Appl.* **2020**, *7*, 279–301. [CrossRef]
99. Cao, Y.; Ji, R.; Ji, L.; Lei, G.; Wang, H.; Shao, X. l^2-MPTCP: A Learning-Driven Latency-Aware Multipath Transport Scheme for Industrial Internet Applications. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8456–8466. [CrossRef]
100. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*; Springer: Boston, MA, USA, 1992; pp. 5–32.
101. Naeem, M.; Rizvi, S.T.H.; Coronato, A. A gentle introduction to reinforcement learning and its application in different fields. *IEEE Access* **2020**, *8*, 209320–209344. [CrossRef]
102. Coulom, R. Efficient selectivity and backup operators in Monte-Carlo tree search. In Proceedings of the Computers and Games: 5th International Conference, CG 2006, Turin, Italy, 29–31 May 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 72–83.
103. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef]
104. Ho, J.; Ermon, S. Generative adversarial imitation learning. In Proceedings of the Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 5–10 December 2016.
105. Gao, Z.; Jiang, L.; Xia, X.; Lo, D.; Grundy, J. Checking Smart Contracts With Structural Code Embedding. *IEEE Trans. Softw. Eng.* **2021**, *47*, 2874–2891. [CrossRef]
106. Hao, X.; Ren, W.; Zheng, W.; Zhu, T. SCScan: A SVM-Based Scanning System for Vulnerabilities in Blockchain Smart Contracts. In Proceedings of the IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Guangzhou, China, 29 December–1 January 2020; pp. 1598–1605.
107. Lou, Y.; Zhang, Y.; Chen, S. Ponzi Contracts Detection Based on Improved Convolutional Neural Network. In Proceedings of the IEEE International Conference on Services Computing (SCC), Beijing, China, 7–11 November 2020; pp. 353–360.
108. Qian, P.; Liu, Z.; He, Q.; Zimmermann, R.; Wang, X. Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models. *IEEE Access* **2020**, *8*, 19685–19695. [CrossRef]
109. Hara, K.; Takahashi, T.; Ishimaki, M.; Omote, K. Machine-learning Approach using Solidity Bytecode for Smart-contract Honeypot Detection in the Ethereum. In Proceedings of the IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), Hainan, China, 6–10 December 2021; pp. 652–659.
110. Mi, F.; Wang, Z.; Zhao, C.; Guo, J.; Ahmed, F.; Khan, L. VSCL: Automating Vulnerability Detection in Smart Contracts with Deep Learning. In Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Sydney, Australia, 3–6 May 2021; pp. 1–9.
111. Wang, B.; Chu, H.; Zhang, P.; Dong, H. Smart Contract Vulnerability Detection Using Code Representation Fusion. In Proceedings of the 28th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taiwan, 6–9 December 2021; pp. 564–565.
112. Yu, X.; Zhao, H.; Hou, B.; Ying, Z.; Wu, B. DeeSCVHunter: A Deep Learning-Based Framework for Smart Contract Vulnerability Detection. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
113. Zhang, Y.; Kang, S.; Dai, W.; Chen, S.; Zhu, J. Code Will Speak: Early detection of Ponzi Smart Contracts on Ethereum. In Proceedings of the 2021 IEEE International Conference on Services Computing (SCC), Chicago, IL, USA, 5–10 September 2021; pp. 301–308.
114. Andrijasa, M.F.; Ismail, S.A.; Ahmad, N. Towards Automatic Exploit Generation for Identifying Re-Entrancy Attacks on Cross-Contract. In Proceedings of the IEEE Symposium on Future Telecommunication Technologies (SOFTT), Johor Baharu, Malaysia, 14–16 November 2022; pp. 15–20.
115. Ashizawa, N.; Yanai, N.; Cruz, J.P.; Okamura, S. Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts. *Blockchain Res. Appl.* **2022**, *3*, 100101. [CrossRef]

116. Gupta, R.; Patel, M.M.; Shukla, A.; Tanwar, S. Deep learning-based malicious smart contract detection scheme for internet of things environment. *Comput. Electr. Eng.* **2022**, *97*, 107583. [CrossRef]
117. Hu, H.; Bai, Q.; Xu, Y. Scsguard: Deep scam detection for ethereum smart contracts. In Proceedings of the IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), New York, NY, USA, 2–5 May 2022; pp. 1–6.
118. Hwang, S.-J.; Choi, S.-H.; Shin, J.; Choi, Y.-H. CodeNet: Code-Targeted Convolutional Neural Network Architecture for Smart Contract Vulnerability Detection. *IEEE Access* **2022**, *10*, 32595–32607. [CrossRef]
119. Li, N.; Liu, Y.; Li, L.; Wang, Y. Smart Contract Vulnerability Detection Based on Deep and Cross Network. In Proceedings of the 2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA), Changchun, China, 20–22 May 2022; pp. 533–536.
120. Liu, L.; Tsai, W.-T.; Bhuiyan, M.Z.A.; Peng, H.; Liu, M. Blockchain-enabled fraud discovery through abnormal smart contract detection on Ethereum. *Future Gener. Comput. Syst.* **2022**, *128*, 158–166. [CrossRef]
121. Nguyen, H.H.; Nguyen, N.M.; Xie, C.; Ahmadi, Z.; Kudendo, D.; Doan, T.N.; Jiang, L. MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities. In Proceedings of the IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA), Online, 13–16 October 2022; pp. 1–10.
122. Shakya, S.; Mukherjee, A.; Halder, R.; Maiti, A. Chaturvedi, SmartMixModel: Machine Learning-based Vulnerability Detection of Solidity Smart Contracts. In Proceedings of the 2022 IEEE International Conference on Blockchain (Blockchain), Espoo, Finland, 22–25 August 2022; pp. 37–44.
123. Wang, Z.; Zheng, Q.; Sun, Y. GVD-net: Graph embedding-based Machine Learning Model for Smart Contract Vulnerability Detection. In Proceedings of the International Conference on Algorithms, Data Mining, and Information Technology (ADMIT), Xi'an, China, 23–25 September 2022; pp. 99–103.
124. Wu, Z.; Li, S.; Wang, B.; Liu, T.; Zhu, Y.; Zhu, C.; Hu, M. Detecting Vulnerabilities in Ethereum Smart Contracts with Deep Learning. In Proceedings of the 4th International Conference on Data Intelligence and Security (ICDIS), Shenzhen, China, 24–26 August 2022; pp. 55–60.
125. Xu, G.; Liu, L.; Zhou, Z. Reentrancy Vulnerability Detection of Smart Contract Based on Bidirectional Sequential Neural Network with Hierarchical Attention Mechanism. In Proceedings of the 2022 International Conference on Blockchain Technology and Information Security (ICBCTIS), Huaihua, China, 15–17 July 2022; pp. 56–59.
126. Zhang, L.; Wang, J.; Wang, W.; Jin, Z.; Su, Y.; Chen, H. Smart contract vulnerability detection combined with multi-objective detection. *Comput. Netw.* **2022**, *217*, 109289. [CrossRef]
127. Zheng, Z.; Chen, W.; Zhong, Z.; Chen, Z.; Lu, Y. Securing the ethereum from smart ponzi schemes: Identification using static features. *ACM Trans. Softw. Eng. Methodol.* **2022**. [CrossRef]
128. Zhou, Q.; Zheng, K.; Zhang, K.; Hou, L.; Wang, X. Vulnerability Analysis of Smart Contract for Blockchain-Based IoT Applications: A Machine Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 24695–24707. [CrossRef]
129. Cai, J.; Li, B.; Zhang, J.; Sun, X.; Chen, B. Combine sliced joint graph with graph neural networks for smart contract vulnerability detection. *J. Syst. Softw.* **2023**, *195*, 111550. [CrossRef]
130. Jiang, F.; Cao, Y.; Xiao, J.; Yi, H.; Lei, G.; Liu, M.; Deng, S.; Wang, H. VDDL: A deep learning-based vulnerability detection model for smart contracts. In Proceedings of the International Conference on Machine Learning for Cyber Security, Nadi, Fiji, 2–4 December 2023; pp. 72–86.
131. Jie, W.; Chen, Q.; Wang, J.; Koe, A.S.V.; Li, J.; Huang, P.; Wu, Y.; Wang, Y. A Novel Extended Multimodal AI Framework towards Vulnerability Detection in Smart Contracts. *Inf. Sci.* **2023**, *636*, 118907. [CrossRef]
132. Liu, Z.; Qian, P.; Wang, X.; Zhuang, Y.; Qiu, L.; Wang, X. Combining Graph Neural Networks With Expert Knowledge for Smart Contract Vulnerability Detection. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 1296–1310. [CrossRef]
133. Su, J.; Dai, H.-N.; Zhao, L.; Zheng, Z.; Luo, X. 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, Rochester, MI, USA, 10–14 October 2023.
134. Sun, X.; Tu, L.; Zhang, J.; Cai, J.; Li, B.; Wang, Y. ASSBert: Active and semi-supervised bert for smart contract vulnerability detection. *J. Inf. Secur. Appl.* **2023**, *73*, 103423. [CrossRef]
135. Zhang, Z.; Lei, Y.; Yan, M.; Yu, Y.; Chen, J.; Wang, S.; Mao, X. Reentrancy vulnerability detection and localization: A deep learning based two-phase approach. In Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, Rochester, MI, USA, 10–14 October 2023.