electronics

Review

Automated Repair of Smart Contract Vulnerabilities: A Systematic Literature Review

Rasoul Kiani and Victor S. Sheng *

Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA; rasoul.kiani87@yahoo.com
* Correspondence: victor.sheng@ttu.edu

Abstract: The substantial value held by smart contracts (SCs) makes them an enticing target for malicious attacks. The process of fixing vulnerabilities in SCs is intricate, primarily due to the immutability of blockchain technology. This research paper introduces a systematic literature review (SLR) that evaluates rectification systems designed to patch vulnerabilities in SCs. Following the guidelines set forth by the PRISMA statement, this SLR meticulously reviews a total of 31 papers. In this context, we classify recently published SC automated repair frameworks based on their methodologies for automatic program repair (APR), rewriting strategies, and tools for vulnerability detection. We argue that automated patching enhances the reliability and adoption of SCs, thereby allowing developers to promptly address identified vulnerabilities. Furthermore, existing automated repair tools are capable of addressing only a restricted range of vulnerabilities, and in some cases, patches may not be effective in preventing the targeted vulnerabilities. Another key point that should be taken into account is the simplicity of the patch and the gas consumption of the modified program. Alternatively, large language models (LLMs) have opened new avenues for automatic patch generation, and their performance can be improved by innovative methodologies.

Keywords: smart contract; APR; vulnerability

Citation: Kiani, R.; Sheng, V.S. Automated Repair of Smart Contract Vulnerabilities: A Systematic Literature Review. *Electronics* **2024**, *13*, 3942. https://doi.org/10.3390/electronics13193942

Academic Editors: Boni García, Mario Munoz-Organero, Patricia Callejo and Miguel Ángel Hombrados

Received: 8 September 2024
Revised: 28 September 2024
Accepted: 4 October 2024
Published: 6 October 2024

1. Introduction

A smart contract (SC) is defined as a digital agreement that executes on a blockchain network [1,2]. Employing SCs permits the automatic fulfillment of contract terms, thus enhancing the efficiency of decentralized applications [1,3]. These digital contracts closely resemble the structure of If–Then statements prevalent in diverse programming environments [4]. Blockchain is vulnerable to flaws in the design and implementation of SCs. Vulnerabilities existing int the SC's source code also put blockchain at risk [2].

The available SC analysis tools are restricted in their ability to identify vulnerabilities and lack the capability to address them through patching. To mitigate this problem, researchers are turning to rectification systems as a potential alternative approach. The immutability of the blockchain makes it impossible to utilize traditional program repair techniques for fixing issues in SCs [1]. In this context, the key question is the following: What makes the existence of unfixed bugs in SCs critical? The response from Yu et al. [5] to the posed question includes the following points: (i) the overall state of SCs is visible to everyone, (ii) any patch developed for a vulnerable SC must not only rectify the identified vulnerabilities but also take into account the gas consumption of the modified program, and (iii) the quality of the patch produced for a vulnerable SC is a critical design consideration, particularly since SCs are predominantly utilized in commercial transactions.

We observed that a significant majority of proposed frameworks, surveys, and systematic reviews focus primarily on vulnerability detection tools [4]. However, researchers have almost overlooked the issue of vulnerability correction [2]. We categorize automated program repair (APR) methodologies within SCs into five different perspectives: search-based, constraint-based, template-based, learning-based, and large language model (LLM)-based approaches. Furthermore, we provide a taxonomy that organizes these methods according to their respective patching levels, specifically bytecode-based and source code-based strategies. Additionally, we explore the types of vulnerability detection tools that are adopted by each repair framework. Driven by the need to overcome these challenges, we arranged an SLR about SC automated repair.

In summary, the novel and significant contributions of this paper are enumerated as follows:

- The focus is on cutting-edge solutions that have been applied for vulnerability correction in SCs. In this respect, 31 journal articles and conference papers published between 2020 and 21 June 2024 will be explored.
- We classify automated repair frameworks under three different perspectives: the approaches related to APR, the rewriting strategies employed, and the detection tools utilized.
- We highlight a number of contemporary open challenges intended to tackle substantial issues that occur when different frameworks are applied to the correction of vulnerabilities.

The remainder of this paper is organized as follows: Preliminaries are introduced in Section 2. Section 3 outlines the key findings of the related studies. In Section 4, the research methodology covers our paper selection procedure. The classification of the selected papers is represented in Section 5. Section 6 provides an analytical comparison. Sections 7 and 8 ultimately result in discussions and conclusions.

2. Preliminaries

This section introduces terminology, such as blockchain, Ethereum, SCs, vulnerability, and automated SC vulnerability correction necessary for this paper. The results of our study reveal that the following expressions are interchangeable: repair (n/v), patch (n/v), correction (n), fix (n/v), rectify (v), and mitigation (n), where *n* stands for a noun and *v* represents a verb.

2.1. Blockchain

The blockchain is created by sequentially linking data blocks based on their timestamps. Blockchain technology has progressed through three different stages of development. After successfully completing the initial 1.0 stage, the project is now in the process of transitioning to blockchain 2.0, where SCs will be utilized [6–8].

2.2. Ethereum

Ethereum stands out as the leading blockchain platform that facilitates SCs. It promotes the execution and invocation environments of SCs via a Turing-complete mechanism known as the Ethereum virtual machine (EVM) [9,10].

2.3. Smart Contract (SC)

The concept of SCs was initially introduced by Szabo [11], emphasizing their ability to simplify the process of executing contracts using protocols and user interfaces. Once the predetermined conditions are met, the blockchain is updated and executed. After executing a contract associated with the transaction, the transaction result becomes immutable and irreversible. A smart contract encompasses a collection of state variables and executable functions. SC technology relies on three key components: the platform, the programming language, and the execution environment [6,12].

Blockchain SC execution mechanism can be divided into on-chain, off-chain, and hybrid methods [13]. To clarify, consider blockchain as a cloud storage system that includes two essential parts: a public section and a private section. On-chain transactions are comparable to the public cloud, being visible to all users, while off-chain transactions resemble the private cloud, where the data are kept confidential and not accessible to the public. An alternative strategy for improving the execution of on-chain SCs is off-chain execution. This method enables the handling of tasks associated with SCs to occur outside the blockchain environment. Isra et al. [14] define on-chain, off-chain, and hybrid mechanisms as follows:

- An on-chain execution is defined as any operation that is replicated across all participants within the blockchain network.
- Off-chain refers to the process of assigning the execution or operation that influences the ultimate state of the blockchain ledger to a specific group of nodes.
- Hybrid execution refers to a process that integrates both off-chain and on-chain operations in its execution framework.

Table 1 details cutting-edge strategies that follow SC execution mechanisms.

Table 1. SC execution mechanism frameworks.

Mechanism	Execution Type	Detail
SMACS [15]	off-chain	SMACS serves as an off-chain framework that delivers precise access control guidelines for SCs, characterized by its dynamic extensibility and reduced on-chain overhead. In order to activate SMACS, the owner of the SC is required to amend the contract to incorporate the verification of SMACS tokens for each method that can be accessed externally. Nonetheless, it breaches the immutability principle by allowing the contract owner to make dynamic updates to the rules. In addition, the SMACS cannot provide protection for contracts that were deployed without a pre-established upgrade mechanism.
POSE [16]	off-chain	POSE offers robust security assurances, even in scenarios where a significant number of participants are compromised. The purpose of POSE is to permit a collection of users to operate an intricate SC on several systems that are enabled with Trusted Execution Environments (TEEs). This framework, however, is independent of any specific TEE. This off-chain-based strategy functions by randomly choosing a group of TEEs to carry out each SC, designating one enclave as the executor while the remaining enclaves serve as watchdogs.
Chen et al.'s model [17]	hybrid	The research conducted by Chen et al. [17] presents a novel concurrent execution strategy that merges off-chain execution with the on-chain concurrent execution framework. This integration allows a blockchain system to boost its performance without sacrificing security and decentralization. By organizing transaction scheduling information in relation to execution-related data, this framework improves the efficiency of both information dissemination and execution among nodes.
SRP [18]	hybrid	SRP provides real-time protection for deployed SCs against attacks, ensuring that the performance of the underlying blockchain remains unaffected. This framework outlines a protocol tailored for interoperability in off-chain runtime verification. The SRP exhibits enhanced service time and throughput when contrasted with an on-chain-only mechanism, especially in scenarios involving escalating workloads.
Twin-Ledger [19]	off-chain	Twin-Ledger architecture introduces an innovative method to address the challenges of throughput without expanding block size and substantial space demands in public ledgers. This framework enables the integration of any consensus mechanism, but it particularly leverages the Proof-of-Work consensus to achieve scalability and decentralization without sacrificing security.
ICOE [20]	off-chain	ICOE serves as an off-chain execution paradigm aimed at addressing the scalability difficulties linked to SC-based industrial applications within blockchain environments. This framework allows the invoked SC to operate only within its specific group, ensuring that the execution result is returned solely to the invoked SC. Nevertheless, irrespective of the volume of data stored in SCs, the ICOE system transfers SCs that are initially stored on-chain to various off-chain consensus groups. This significantly alleviates the storage demands on the on-chain blockchain. Furthermore, the ICOE system is capable of supporting a wide range of applications based on smart contracts.
Isras et al.'s model [14]	off-chain	The research conducted by Isra et al. [14] presents a queuing model focused on off-chain runtime verification and the process of block generation. This model is capable of efficiently and flexibly representing the non-deterministic nature of blockchain systems, allowing for the estimation of both the number of transactions in the pool and their respective waiting times.

2.4. Vulnerability

In general, SC vulnerabilities can be categorized into three categories [5,6,12]: (i) vulnerabilities at the blockchain level, (ii) vulnerabilities at the Ethereum virtual machine level, and (iii) vulnerabilities at the source code level (high-level languages like Solidity. The most discussed Ethereum SC vulnerabilities include timestamp dependency [21,22], reentrancy [21–24], Transaction Ordering Dependency [21,22], tx.origin [21–24], BlockhashBlock Number [21,22], Gas-Related Issues [21,22], delegate call [21,24], Arithmetic UnderflowOverflow [21,23,24], unchecked call [23], Self-Destruct [21,23,24], access control [23], and denial of service [23].

2.5. Automated Smart Contract Vulnerability Correction

The automated SC repair issue, as described by Yu et al. [5], involves the challenge of developing an algorithm that can effectively address vulnerabilities in a SC. This algorithm must take into account key factors, including the initial vulnerable SC, a list of identified vulnerabilities, a test suite, and a specified maximum gas usage limit. The ultimate goal is to generate a new contract that closely resembles the original one, with all vulnerabilities rectified and successfully passing all tests while ensuring that the gas consumption of feasible execution paths does not exceed the predetermined limit.

3. Related Work

This section reviews eight surveys and SLRs published from 2020 to 2024, specifically, fixing vulnerable patterns in SCs. Then, we discuss the advantages and limitations of each study. Table 2 details review papers related to fixing vulnerabilities in SCs. Table 3 refers to research questions, followed by recently published survey papers.

Table 2. Overview of related surveys and their coverage based on SC automated repair.

Reference	Environment	Review Type	Year of Pub	Selection Process	Taxonomy	Year Covered
[25]	Solidity	empirical review	2020	semi-clear [1]	-	14 December 2018–31 March 2019
[26]	SC formal specification and verification	survey	2021	clear [2]	✓	September 2014–June 2020
[27]	Formal verification for Solidity SCs	survey	2021	semi-clear [3]	✓	2008–2018 [4]
[28]	Bugs	SLR	2023	not clear [5]	✓	2021–2022 [6]
[1]	DeFi [7] security	empirical review	2023	not clear [5]	-	2020–2021
[29]	Off-chain and on-chain repair	SLR	2023	clear [2]	✓	2015–2023
[30]	Bugs	empirical review	2023	clear [2]	✓	2016–2021
[3]	APR	survey	2024	not clear [5]	-	2019–2023 [4]

[1] The identification methodology is clear, while the exclusion, eligibility, and inclusion methodology are not clear. [2] Search queries, keywords, and organization of the article are clear. [3] The scientific databases and keywords are clear. [4] It is not mentioned directly, but the survey's main references were published between these years. [5] The identification, exclusion, eligibility, and inclusion methodology are not clear. [6] Covering 516 unique real-world SC vulnerabilities in years 2021–2022. [7] Decentralized Finance.

Table 3. Research questions followed by survey papers.

Reference	Research Questions
[25]	1: Does the Solidity language become more secure by fixing known vulnerabilities? 2: Does the Solidity compiler get security patches for the known vulnerabilities? 3: Would Solidity developers take advantage of the repaired Solidity compilers? 4: Are the compiler patches effective enough to prevent their target vulnerabilities? 5: Do developers actually make bugs and errors due to such vulnerabilities?
[26]	1: What are the formal techniques used for modeling, specification, and verification of SCs? 2: What are the common formal requirements specified and verified by these techniques? 3: What are the challenges introduced by SC and blockchain environment in formalizing and verifying SCs? 4: What are the current limitations in SC formal specification and verification and what research directions may be taken to overcome them?
[28]	1: What kinds of exploitable bugs are machine auditable by existing tools? 2: How many real-world exploitable bugs are machine auditable? 3: How difficult is it to audit exploitable bugs? 4: What are the root causes, categories, and distributions of machine unauditable bugs? 5: What are the symptoms and fixes of machine unauditable bugs?
[1]	1: How do traditional SC vulnerability detection tools perform when applied to DeFi protocols? 2: What is the performance of state-of-the-art tools in detecting DeFi attacks or vulnerabilities?
[29]	1: What are the existing vulnerability defense methods? 2: What are the strengths and limitations of those repair methods? 3: What vulnerabilities are covered by those vulnerability repair methods?
[30]	1: What types of files are involved when fixing bugs? 2: How many Solidity files are modified during a fix? 3: How many Solidity files are necessary to be added or deleted to fix bugs? 4: What is the most common fix operation during bug fixes? 5: How many element kinds are modified in a Solidity file during bug fixes? 6: How many fix actions are taken to a Solidity file during bug fixes?

The survey [25] reveals the gap between theory and practice in security patches in Solidity, the most popular programming language used by Ethereum SC developers. The method of article selection has not been well-demonstrated in this research. However, the search domain for this article is clear. The scholars recommended five research questions focusing on the relationship between Solidity and vulnerability correction in SCs. They found that 7 out of 41 known vulnerabilities are patched.

In another study, a survey [26] reviews major trends, challenges, and future directions in the formal specification and verification of SCs. The researchers effectively indicated their article selection method. The scope of their article searches spans from September 2014 to June 2020. The authors collected papers from scientific databases, including Google Scholar, IEEE, Springer, and ACM. They used the following keywords for the search: formal, verification, specification, modeling, smart contract properties, smart contract temporal properties, and temporal properties. According to the authors, the use of contract-level models and specifications in conjunction with model checking is a common method for reasoning about the functional correctness of SCs in diverse domains.

Furthermore, the survey [27] explores formal verification for Solidity SCs. The authors classify and assess gathered research studies according to two key criteria: (i) the verification methods employed and (ii) the vulnerabilities that are the main focus of the research. Specifically, they categorize two families of formal verification methods, those that rely on theorem proving and those that rely on model checking. Their article search domain is not clear. The researchers gathered articles from academic repositories, such as Google Scholar and DBPL computer science bibliography. The search utilizes the subsequent keywords: smart contract, formal verification, Solidity, and Ethereum.

Likewise, a systematic review [28] categorizes SC bugs that cannot be detected by existing tools and studies their underlying causes, distributions, impacts, consequences, and methods for repair. The article illustrates the symptoms and repair strategies of price oracle manipulation, Erroneous Accounting, and privilege escalation bugs through the use of real examples. It includes ten findings. The method of article selection in their study has not been clearly revealed. Their article search domain spans from 2021 to 2022.

An empirical review [1] explores the advancements achieved in SC and DeFi security, focusing on the detection of vulnerabilities and the implementation of automated repair mechanisms. The criteria for selecting articles are not transparent. It reviews eight automated repair tools for SCs and DeFi protocols, which were proposed between 2020 and 2022. According to the survey, contemporary automated repair tools are able to address only certain types of vulnerabilities and face challenges in dealing with complex DeFi protocols.

The SLR [29] provides insights into data sources, detection methods, and repair strategies related to vulnerabilities in SCs. The researchers have successfully detailed their approach to article selection. Their search criteria cover the years from 2015 to 2023. However, the research is restricted to five vulnerability repair frameworks that were released from 2020 to 2021. The study underscores the importance of developing off-chain repair solutions that offer superior performance to maintain the security of contracts. Conversely, there is a necessity for the advancement of on-chain repair technologies to facilitate real-time updates of the patch contract, thereby enhancing the security of the contract.

Another empirical study [30] explores historical bug fixes from 46 real-world Solidity SCs projects. This research offers practical recommendations for enhancing existing methodologies for addressing bugs in Solidity SCs, focusing on three key areas: automated repair techniques, analytical tools, and the role of Solidity developers. The criteria for selecting projects are transparent.

The SLR [3] examines existing methods for detecting and repairing contract vulnerabilities. It explores six papers published between 2019 and 2023. The research emphasizes the increasing shift towards using Generative AI to fix vulnerabilities in SCs. There is a lack of transparency in the selection criteria for articles. Moreover, the scholars do not introduce a taxonomy regarding automated repair of SC vulnerabilities.

In summary, reviewing previous articles, the ensuing shortcomings were discovered: (1) The survey articles did not include the newly proposed APR tools for SCs. (2) Survey papers have neglected the taxonomy of vulnerability correction tools, which relies on APR approaches, patching levels, and detection tools. In light of the aforementioned limitations, this SLR has been introduced to address the issues identified in prior studies.

4. Methodology

In our SLR on the correction of vulnerabilities in SCs, we adhered to the guidelines set forth by Moher et al. [31]. Here, we will detail the selection criteria for the analyzed papers, the methodologies adopted for their preparation, and provide an overview of the research outcomes.

4.1. Research Questions (RQs)

Despite advancements, there are still challenges to overcome in developing automatic patch generation systems for SCs, as indicated by the study of review articles and SLRs. The present investigations are entirely valid and significant to the relevant field. In this context, three questions were formulated, and they will be addressed through both analytical and graphical approaches in the upcoming sections.

- RQ1: What recent trends automatically generate security patches for vulnerable SCs?
- RQ2: What types of vulnerabilities are fixed by novel rectification frameworks?
- RQ3: What types of vulnerability detection tools are employed by automated repair frameworks when addressing SCs?

4.2. Selection of Primary Studies

On 21 June 2024, the search was conducted. The platforms used for the search were IEEE, ACM, Springer, ScienceDirect, Wiley, Taylor and Francis, and online archives such as arXiv. The search covers title, abstract, and keywords. To search for primary studies, we have identified the following keywords: "smart contract" AND "vulnerability" AND ("repair" OR "patch" OR "correction" OR "fix" OR "rectify" OR "mitigation").

Each paper was scrutinized based on its title, abstract, introduction, and conclusions to determine its compliance with the established inclusion criteria. Following this, an evaluation of the remaining sections of each document was conducted to pinpoint key contributions and unresolved issues.

4.3. Inclusion and Exclusion Criteria

This SLR is centered on journal articles, international conference proceedings, and symposiums that have been published from 2020 to 21 June 2024. Repetitive articles and various other forms of literature such as books, surveys, empirical studies, critical articles, technical reports, and master's theses were excluded.

4.4. Selection Results

Considering the keywords, the following results have been obtained, divided by platform:

- IEEE: 38 results.
- ScienceDirect: 13 results.
- Springer: 15 results.
- ACM: 41 results.
- Wiley: 8 results.
- Taylor and Francis: 3 results.
- Online archives: 8.

Subsequently, the duplicated studies were taken out. Then, papers that did not meet the exclusion criteria were excluded. Ultimately, 31 papers were left for analysis. The methodology for including and excluding research articles is outlined in Figure 1.

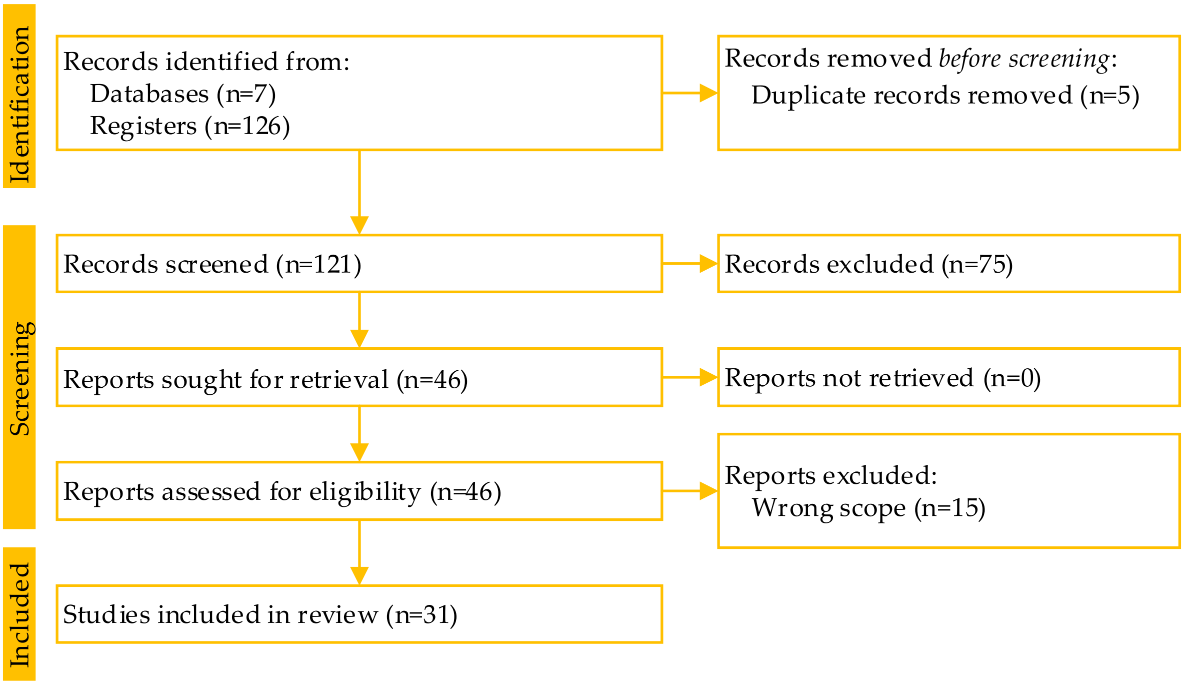
Identification
Records identified from:
Databases (n=7)
Registers (n=126)
Records removed *before screening*:
Duplicate records removed (n=5)
Screening
Records screened (n=121)
Records excluded (n=75)
Reports sought for retrieval (n=46)
Reports not retrieved (n=0)
Reports assessed for eligibility (n=46)
Reports excluded:
Wrong scope (n=15)
Included
Studies included in review (n=31)

Figure 1. Literature selection process based on PRISMA flowchart on three levels.

Furthermore, we chart the number of papers released annually to expose temporal tendencies (see Figure 2).

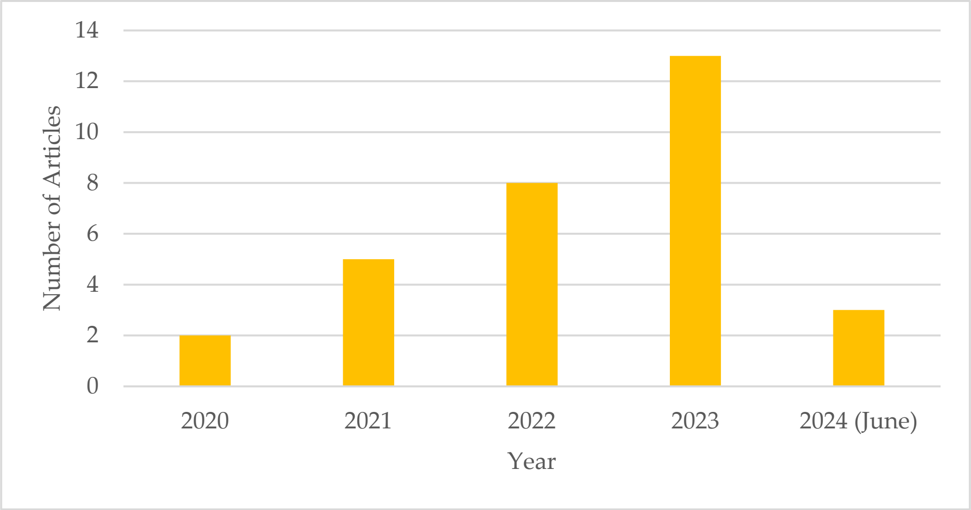

Figure 2. The number of covered papers published between 2020 and June 2024.

5. Taxonomy of Fixing Techniques for SC Vulnerabilities

This section outlines the evolution of APR techniques in SCs through five different perspectives: search-based, constraint-based, template-based, learning-based, and LLM-based methodologies. We also present a taxonomy that categorizes repairing frameworks according to their patching levels, specifically bytecode-based and source code-based strategies. Furthermore, we investigate the types of vulnerability detection tools that are employed by each repair framework.

5.1. Search-Based Approach

Heuristic-based solutions, often called search-based methods, are grounded in the core idea of searching through a predefined patch space to pinpoint the appropriate patch [32]. Existing search-based APR frameworks leverage the advantages of both bytecode-based and source code-based rewriting strategies.

5.1.1. Bytecode-Level Rewriting Strategy

Hou et al. [33] introduce HermHD, an automated tool designed to enhance security by utilizing six distinct obfuscation patterns, which enable the rewriting of an SC's bytecode while preserving its original functionality. It protects Ethereum SCs against reverse engineering. SC code obfuscation is a method that can effectively hide the business logic, semantics, and other pertinent information associated with the contract. This approach not only ensures the privacy of the SC but also compromises the efficacy of analysis tools, resulting in an inability to accurately analyze the semantics of the SC.

5.1.2. Source Code-Level Rewriting Strategy

Yu et al. [5], Nassirzadeh et al. [34], and Tolmach et al. [26] propose a source code-level rewriting strategy to automate the repair of SCs based on a search method. SCRepair [5] is a gas-aware and general-purpose framework that takes into account the gas consumption of the potential patches. Gas Gauge [34] includes three essential components: the detection phase, the identification phase, and the correction phase. The correction phase component employs static analysis alongside run-time verification to estimate the maximum loop bounds that align with permissible gas usage. Nonetheless, SCRepair and Gas Gauge focus on static analysis tools to identify specific vulnerabilities within SCs and select suitable fixing patterns; DeFinery [26] adopts a semantic methodology. Notably, the patch generation component of DeFinery is established on the SCRepair framework.

5.2. Constraint-Based Approach

The primary principle underlying constraint-based methods, referred to as semantic constraint approaches, influences the repair process by generating a defined set of constraint specifications [32]. Existing constraint-based APR frameworks leverage the advantages of source code-based rewriting strategies.

The current constraint-based methodologies include the approach by Ren et al. [35] and the SymlogRepair framework [36]. Ren et al.'s model [35] includes two primary components: security-reinforced code suggestion and security-oriented code validation. They identify and rectify specific security vulnerabilities utilizing Abstract Syntax Tree (AST) and Datalog. Their security-reinforced code suggestion benefits from the functionality of a long short-term memory (LSTM) network. The SymlogRepair framework [36] aims to combine program repair with Datalog-based analysis. However, to effectively support a new type of bug in this paradigm, it is imperative to create a Datalog program that facilitates the identification of this particular class of bugs.

5.3. Template-Based Approach

Template-oriented solutions, rule-based, transformation-based, or pattern-based, employ a predetermined program fix template to generate repair patches. These fix templates can be manually extracted or automatically derived through mining techniques [32]. Existing template-based APR frameworks leverage the advantages of bytecode-based and source code-based rewriting strategies.

5.3.1. Bytecode-Level Rewriting Strategy

The works of Zhang et al. [37], Rodler et al. [38], Jin et al. [39], Torres et al. [40], Guo [41], Feng et al. [42], Shi et al. [43], and Huang et al. [44] introduce template-driven methods for correcting vulnerabilities, which employ a rewriting scheme at the bytecode level.

Zhang et al. [37] developed a gas-friendly bytecode rectification system called SmartShield to address three prevalent security-related bugs. By fixing insecure instances within each vulnerable SC, this framework ensures the security of the EVM bytecode during the contract's ultimate deployment. They utilize a semantic-preserving transformation technique to compile each insecure contract into a secure bytecode version. In this strategy, symbolic execution and abstract interpretation techniques are utilized to verify the correctness of the rectified contracts. In contrast to SmartShield, EVMPatch [38] operates when the vulnerability is situated within a single bytecode basic block, struggling to address vulnerabilities across different basic blocks. However, it is not entirely automated, which means that the developer is obligated to resolve the bug manually. SmartShield and EVMPatch implement supplementary runtime checks, potentially resulting in considerable performance costs. The fundamental principle behind Aroc [39] is to utilize separate SCs equipped with patches to block malicious transactions in advance. However, the owner is required to execute a designated transaction through the improved EVM to envelop the contract with the necessary patch. This framework has higher runtime overheads than EVMPatch. Elysium proposed by Torres et al. [40] is another SC vulnerability correction scheme at the bytecode level. This framework generally decreases the costs associated with runtime, particularly regarding transaction expenses. Elysium presents a patching strategy that is context-aware, merging template-based and semantic-based methods to develop customized patches for SCs.

Guo [41] introduces an automatic patch generation system known as APG, integrated within the SolSaviour framework [45]. This system addresses the security issues associated with patches implemented by SolSaviour and improves the protective capabilities of SolSaviour for SCs. This paradigm is formed by the integration of two existing frameworks, which are Slither and EVMPatch [38]. While the security measures are improved, the associated computing costs are elevated. The research conducted by Feng et al. [42] benefits from the BiLSTM model to detect reentrancy vulnerabilities and a bytecode rewriting strategy. This method distinguishes itself from earlier approaches by employing the synthetic minority over-sampling technique (SMOTE), which improves the dataset by incorporating additional samples from minority classes. The work of Shi et al. [43] unveils EtherEditor, which tackles the challenges related to widely recognized automated repair tools such as SCRepair [5], SmartShield [37], EVMPatch [38], and Elysium [40]. Huang et al. [44] introduce a novel framework for repairing SCs, known as ReenRepair, which is aimed specifically at locating reentrancy vulnerabilities and providing semantically equivalent solutions for their repair. In this context, the authors outline two scenarios of false positives that are not influenced by reentrancy attacks. They subsequently develop a model that incorporates read–write dependencies and path connectivity to mitigate false positives in the localization of reentrancy. Furthermore, they adopt two gas-optimized repair templates to tackle reentrancy: the bit-lock template and the reordering template. In terms of repair efficiency, ReenRepair outperforms SCRepair [5].

5.3.2. Source Code-Level Rewriting Strategy

The template-oriented frameworks put forth by Nguyen et al. [46], Thyagarajan et al. [47], Li et al. [45], Giesen et al. [48], Beillahi et al. [49], Antonino et al. [50], Chen et al. [51], Fang et al. [52], XI and Pattabiraman [53], Gao et al. [54] are established on a rewriting strategy that is guided by the source code.

Nguyen et al. [46] conduct the SGuard framework, which follows a runtime verification strategy to correct automatically four kinds of vulnerabilities. The patch code introduced by SGuard might provide inadequate protection, leading to possible changes in the semantics of the original code. In contrast, Reparo [47] is a publicly verifiable layer that can be implemented on any blockchain to facilitate repairs, including the correction of faulty contracts and the elimination of illicit content from the chain. This protocol can be customized to fit any flavor of consensus, including permission systems, without introducing any overhead. SolSaviour [45] is reported to protect deployed SCs from unknown vulnerabilities. This strategy is composed of two fundamental components: voteDestruct and the TEE cluster. The voteDestruct mechanism is integrated into SCs before their deployment, allowing for the destruction of a contract through a voting process. Unlike existing solutions that depend on a trusted third party to redeploy updated contracts, SolSaviour facilitates the efficient migration of contract assets without the need for a trusted intermediary. Nonetheless, SolSaviour cannot extend its defense mechanism to SCs that are already in operation and have been deployed.

Due to the code property Graphs (CPGs), HCC [48] can be applied to diverse SC platforms and programming languages. Nonetheless, HCC faces obstacles in rectifying vulnerabilities present in complex DeFi protocols. This framework addresses and mitigates several false alarm issues characteristic of previous solutions like SGuard. Beillahi et al. [49] conducted a methodology to detect transaction order dependency (TOD) vulnerabilities and rectify them. This work is limited by the fact that Slither does not analyze inlined assembly statements found in the SC code. As a result, it may not identify dependencies between the outcomes of transactions and state variables that can be altered. Antonino et al. [50] put forward a systematic deployment framework, called Trusted Deployer that requires formal verification of contracts before they are established and modified. This model guarantees that the original implementation and all future upgrades will comply with the defined specifications. Trusted Deployer utilizes an off-chain strategy and permits developers to correct the contract before its deployment.

TIPS [51] determines the suitable fix template based on the vulnerability category identified by vulnerability detection tools and generates patches heuristically, utilizing the code alteration actions defined by the selected fix template. It can efficiently produce patches for vulnerable SCs, demonstrating superior performance compared to SCRepair [5]. ContractFix [52] facilitates the migration of statements to address reentrancy vulnerabilities while incurring significantly lower gas costs compared to Sguard [46]. Furthermore, EVMPatch [38] sacrifices the semantics of the source code and necessitates additional data analysis compared to the ContractFix [52] framework. GoHigh [53] mechanism involves the scrutiny of the Abstract Syntax Tree (AST) that corresponds to the SC's source code. This strategy replaces low-level functions with more advanced high-level alternatives. This scheme addresses both the unhandled exception-related vulnerabilities and the use of low-level and obsolete functions vulnerabilities.

Gao et al. [54] put forward SGuard+, a machine learning-based automated approach to vulnerability repair, intending to enhance the effectiveness and efficiency of SGuard in relation to vulnerability detection and repair processes. They develop new repair rules that involve fewer code changes and diminish unnecessary patch code, relying on precise localization to lower gas overhead. This framework employs a binary classification machine learning model, specifically eXtreme Gradient Boosting (XGBT), to identify each vulnerability type at the function level. The training dataset is labeled based on the outputs from the Slither, Securify, and Mythril tools. However, the evaluation of the XGBT of SGuard+ reveals two primary limitations associated with the machine learning approach: (1) The models do not achieve consistent performance for all types of vulnerabilities. (2) A tradeoff is observed between recall and precision in the models.

Notably, the approaches employed by the SmartShield [37], SGuard [46], and Elysium [40] frameworks depend on a singular repair strategy for each category of bugs, leading to a failure in rectifying the diverse patterns of bugs encountered.

5.4. Learning-Based Approach

Machine learning-based approaches establish probabilistic models that analyze the distribution of repair patterns, thereby facilitating improved selective fixing strategies [32]. Existing learning-based APR frameworks leverage the advantages of source code-based rewriting strategies.

Zhou et al. [55], Guo et al. [56], and So and Oh [57] have introduced learning-based frameworks that are constructed upon a strategy of rewriting driven by source code. SmartRep [55] employs source code and partial syntax information to efficiently provide one-line patches for SC repairs, intentionally avoiding the extraction of syntax tree structures and bytecode information. This framework is designed with two encoders and a decoder. Each encoder includes an embedding layer and two LSTM layers, whereas the decoder is structured with an embedding layer and a single LSTM layer. In contrast to SCRepair [5], SmartRep can deliver security code recommendations at a significantly faster pace. Furthermore, SCRepair can rectify four categories of vulnerabilities, while SmartRep can patch eleven types of vulnerabilities.

The work of Guo et al. [56] reveals RLRep, an approach based on reinforcement learning that leverages an agent to recommend repair actions for vulnerable SCs, all without supervision. In this context, they develop a comprehensive list of repair actions to direct the agent in generating effective paths for the necessary repairs. The actions are structured based on the official suggestions for corrections and the mutation operators found in conventional programming languages and Solidity. Furthermore, the authors establish the reward function utilizing compilation, vulnerability detection tools, code similarity, and code entropy, which facilitates the optimization of SC repair recommendations across various metrics. LSTM is employed for encoding and decoding sequences due to its effective performance in code repair tasks. This methodology may effectively resolve the challenge posed by the lack of labeled data in machine learning-driven repair methods. However, the framework offers an estimated 55% accuracy in its repair recommendations for SCs. Furthermore, this method presently lacks the capability to provide repair suggestions for emerging vulnerabilities, which consequently restricts its scalability.

So and Oh [57] introduce SmartFix, which utilizes a "generate-and-verify" strategy. This technique iteratively produces candidate patches while ensuring their correctness by engaging a safety verifier for validation. This framework employs a machine learning-based approach that effectively directs the repair process by utilizing statistical models. These models are constructed through both online and offline techniques.

5.5. Large Language Model (LLM)-Based Approach

These models provide a different avenue for research in APR by harnessing their abilities in code understanding and generation to formulate repairs [32]. Existing LLM-based APR frameworks leverage the advantages of source code-based rewriting strategies.

The research conducted by Napoli and Gatteschi [58], Ibba et al. [59], Jain et al. [60], and Zhang et al. [61] has led to the development of LLM-based frameworks that are founded on a rewriting methodology influenced by source code.

Napoli and Gatteschi [58] evaluate the potential of chat generative pre-trained transformer (ChatGPT) in addressing vulnerabilities within SCs. The results demonstrate that ChatGPT could rectify bugs and vulnerabilities in SCs with an average success rate of 57.1%. This rate increased by 1.4% when a detailed description of the bug was included alongside the SC's source code. Nevertheless, they assess ChatGPT's capability to correct code within widely recognized vulnerable SCs. Moreover, it does not address the rectification of unknown vulnerabilities.

Ibba et al. [59] utilize ChatGPT to repair Solidity SCs automatically. The authors investigate three methods by which ChatGPT can facilitate the automatic repair of SCs: (1) utilizing ChatGPT as a tool for vulnerability detection and APR, (2) employing ChatGPT as an APR tool that incorporates training samples, and (3) leveraging ChatGPT as an APR tool that identifies exposed lines and their corresponding vulnerabilities. The initial two models attained accuracy rates of 53% and 39%, respectively. Despite this, the third model achieved an accuracy rate of 89%. Therefore, ChatGPT is most effectively employed as an APR tool when integrated with vulnerability detection systems.

Jain et al. [60] introduce Two Timin, a tool designed to repair SCs, which leverages the capabilities of two prominent LLMs: GPT-3.5-Turbo and Llama-2-7B. In this context, vulnerabilities are identified through an innovative pipeline that utilizes Slither and a Random Forest classifier. Subsequently, the identified malicious SCs and their associated vulnerabilities serve as parameters for prompts directed at two distinct LLMs. This framework facilitates a more comprehensive repair process and is designed to be flexible in addressing zero-day vulnerabilities. The pre-trained GPT-3.5-Turbo and the fine-tuned Llama-2-7B reduced the overall vulnerability count by 97.5% and 96.7%, respectively. However, there is a lack of clarity regarding the particular vulnerabilities that this framework mitigates. In addition, the source code is unavailable. This framework also serves to repair malicious SCs generated by OpenAI's GPT 3.5 Turbo.

Zhang et al. [61] introduce ACFIX, a solution incorporating both online and offline patches to address access control vulnerabilities in SCs. In this framework, they utilize an enhanced version of GPT-4, resulting in a notable improvement over the standard GPT-4 model. In the offline phase, ACFIX extracts a taxonomy of prevalent role-based access control practices from on-chain contracts, systematically categorizing 49 role permission pairs derived from the most distinctive pairs identified. In the online phase, ACFIX monitors AC-related components throughout the contract and utilizes this contextual information with a Chain-of-Thought pipeline. This approach assists LLMs in determining the most suitable role-permission pair for the specific contract, ultimately leading to the generation of an appropriate patch. In contrast to other LLM-based frameworks, this particular framework has the ability to mitigate a specific vulnerability.

6. Comparison

Section 5 contains a detailed analytical discussion and comparison of the final articles. The research questions are addressed in Section 4.1, and this section serves to answer them.

6.1. RQ1: What Recent Trends Automatically Generate Security Patches for Vulnerable SCs?

Table 4 delineates a taxonomy of tools designed for the correction of vulnerabilities in SCs. These tools are classified based on their APR methodologies, which consist of search-based, constraint-based, template-based, learning-based, and LLM-based approaches, along with their rewriting strategies, namely bytecode-level and source code-level. Although

LLM methods can be categorized as a learning model within APR methodologies, we opt to view them as an emerging trend in this discipline.

Table 4. Taxonomy of APR approaches and rewriting strategies.

APR Model and Rewriting Strategy	Availability
Search-based Bytecode-level	
HermHD [33]	https://github.com/ByteCodeMaster/HermHD (7 September 2024)
Search-based Source Code-level	
SCRepair [5]	https://SCRepair-APR.github.io (7 September 2024)
Gas Gauge [34]	https://gasgauge.github.io/ (7 September 2024)
DeFinery [26]	https://sites.google.com/view/ase2022-definery/ (7 September 2024)
Constraint-based Source Code-level	
Ren et al.'s model [35]	https://github.com/FISCO-BCOS/SCStudio (7 September 2024)
SymlogRepair [36]	https://github.com/symlog/symlog (7 September 2024)
Template-based Bytecode-level	
SmartShield [37]	on request
EVMPatch [38]	https://github.com/uni-due-syssec/evmpatch-developer-study (7 September 2024)
Aroc [39]	-
Elysium [40]	https://github.com/christoftorres/Elysium (7 September 2024)
APG [41]	-
Feng et al.'s model [42]	on request
EtherEditor [43]	-
ReenRepair [44]	on request
Template-based Source Code-level	
SGuard [46]	https://github.com/reentrancy/sGuard (7 September 2024)
SolSaviour [45]	-
Reparo [47]	-
HCC [48]	-
TOD [49]	https://github.com/Veneris-Group/TOD-Location-Rectificatio (7 September 2024)
Trusted Deployer [50]	https://github.com/formalblocks/safeevolution (7 September 2024)
TIPS [51]	https://github.com/CVbluecat/TIPS (7 September 2024)
ContractFix [42]	https://github.com/research1132/ContractFix (7 September 2024)
GoHigh [53]	https://github.com/DependableSystemsLab/GoHigh (7 September 2024)
SGuard+ [54]	https://doi.org/10.5281/zenodo.8249340 (7 September 2024)
Learning-based Source Code-level	
SmartRep [55]	https://github.com/AnonymousGithub5/SmartRep (7 September 2024)
RLRep [56]	https://github.com/Anonymous123xx/RLRep (7 September 2024)
SmartFix [57]	https://doi.org/10.5281/zenodo.8256377 (7 September 2024)
LLM-based Source Code-level	
Napoli et al.'s model [58]	https://github.com/enaples/solgpt (7 September 2024)
Ibba et al.'s model [59]	-
Two Timin [60]	-
ACFIX [61]	https://sites.google.com/view/acfixsmartcontract (7 September 2024)

The academic community has extensively investigated template-based APR methodologies, encompassing source code and bytecode rewriting techniques. In a different perspective, the methodologies for APR that rely on constraint-based, learning-based, and LLM-based frameworks have ignored the strategy of bytecode rewriting.

Turning to Figure 3a, the pie chart details the percentage of APR methodologies in SCs. A significant majority of this chart is accounted for template-based methods, and the remaining 42% is used for search-based (12.9%), LLM-based (12.9%), learning-based (9.7%), and constraint-based (6.5%). With respect to Figure 3b, the most prevalent approach for repairing buggy contracts is through the rewriting of source code.

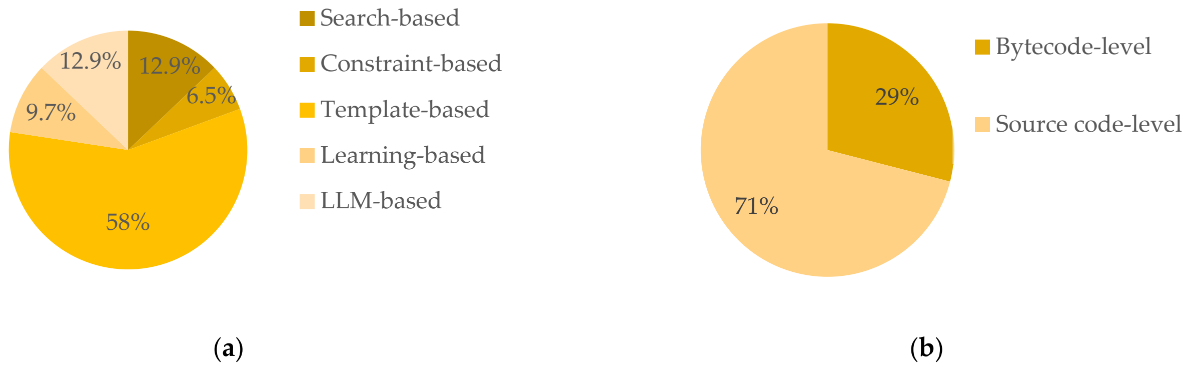

Figure 3. (**a**) Percentage of APR methodologies in SCs and (**b**) percentage of rewriting strategies for vulnerability correction frameworks in SCs.

6.2. RQ2: What Types of Vulnerabilities Are Fixed by Novel Rectification Frameworks?

Table 5 shows different types of vulnerabilities that are fixed by vulnerability correction frameworks.

Table 5. Patching vulnerabilities by vulnerability correction frameworks.

Framework	**# Vulnerabilities**	**Types of Vulnerabilities**
HermHD [33]	1	protect from reverse engineering
SCRepair [5]	4	transaction order dependency, reentrancy, exception disorder, and integer overflow
Gas Gauge [34]	1	out-of-gas denial of services
DeFinery [26]	2	reentrancy and unchecked send
Ren et al.'s model [35]	9	access control, arithmetic, backdoor threats, front running, locked Ether, reentrancy, timestamp dependency, unchecked low calls, and unhandled exception
SymlogRepair [36]	4	access control, unhandled exception, reentrancy, and locked Ether
SmartShield [37]	3	state changes after external calls, missing checks for out-of-bound arithmetic operations, and missing checks for failing external calls
EVMPatch [38]	2	integer over/underflows, and access control errors
Aroc [39]	3	reentrancy, arithmetic bugs, and unchecked low-level checks
Elysium [40]	5	reentrancy, access control, arithmetic, unchecked low-level call, and denial of services
APG [41]	3	malicious voting problems, the patch vulnerabilities, and DeFi unavailability issues
Feng et al.'s model [42]	1	reentrancy
EtherEditor [43]	7	tx.origin authentication, denial of service, reentrancy, unchecked external calls, delegate call to untrusted callee, unprotected SELFDESTRUCTION instruction, and arithmetic over/under flow
ReenRepair [44]	1	reentrancy
SGuard [46]	4	arithmetic (control dependency, data dependency) and reentrancy (intra-function, cross-function)
SolSaviour [45]	9	reentrancy, integer over/underflows, delegate call, denial of service, unchecked return values, front running, timestamp dependency, bad constructor, and unknown bugs
Reparo [47]	2	reentrancy, parity multisig wallet bug
HCC [48]	2	reentrancy, and integer overflows
TOD [49]	1	price gouging transaction order dependency
Trusted Deployer [50]	4	integer overflow and underflow, Nonstandard token interface, wrong operator, and verification error
TIPS [51]	8	unchecked external calls, reentrancy, access control, arithmetic issue, strict balance equality, unmatched type assignment, Inserting a suicide function, and hard-coded address
ContractFix [42]	4	reentrancy, missing input validation, locked Ether, and unhandled exception
GoHigh [53]	4	arithmetic underflow and overflow, reentrancy, denial of service with block gas limit, and unencrypted on-chain data
SGuard+ [54]	5	integer overflow and underflow, unchecked call return value, unprotected self-destruct instruction, reentrancy, and authorization through tx-origin
SmartRep [55]	11	use of deprecated functions, unchecked return value, incorrect inheritance order, illegal coverage, transaction ordering dependence, reentrancy, erroneous visibility, arithmetic issue, missing return statement, authorization through tx.origin, and erroneous variable type
RLRep [56]	5	exception disorder, integer overflow, reentrancy, transaction order dependence, and tx.origin
SmartFix [57]	5	integer over/under-flow, ether-leak, suicidal, reentrancy, and tx.origin
Napoli et al.'s model [58]	-	not mentioned
Ibba et al.'s model [59]	4	reentrancy, denial of service, arithmetic overflows and underflows, and unchecked low-level calls
Two Timin [60]	-	not mentioned
ACFIX [61]	1	role-based access control

6.3. RQ3: What Types of Vulnerability Detection Tools Are Employed by Automated Repair Frameworks When Addressing SCs?

Vulnerability detection tools are classified into two primary classifications: traditional solutions and those utilizing machine learning. The traditional solutions are subdivided into five distinct types: (1) static analysis, (2) symbolic analysis, (3) dynamic analysis, (4) formal verification methods, and (5) fuzzy testing. Meanwhile, machine learning-driven solutions consist of classical models, deep learning models, and ensemble learning models [4]. Apart from the methods already mentioned, some frameworks have explored additional avenues, including (1) a hybrid of traditional techniques and machine learning-based methods and (2) strategies that incorporate internal vulnerability detection tools. Table 6 illustrates the tools that are applied for identifying vulnerabilities in the frameworks that focus on vulnerability mitigation.

Table 6. Vulnerability detection tools employed by automated repair frameworks.

Vulnerability Detection Tools	Automated Repair Frameworks
Traditional Vulnerability Detection Tools	
Mythril and Octopus	HermHD [33]
Oyente and Slither	SCRepair [5]
Slither, Truffle Suite	Gas Gauge [34]
Oyente, Securify, SmartCheck, Pied-Piper, Mythril	Ren et al.'s model [35]
Securify2	SymlogRepair [36]
Securify, Osiris, and Mythril	SmartShield [37]
Osiris, ECF teEther, Oyente, Maian, Sereum, Securify	EVMPatch [38]
Osiris	Aroc [39]
Osiris, Oyente, Mythril	Elysium [40]
Slither	APG [41]
Osiris	EtherEditor [43]
Securify	ReenRepair [44]
Securify, Ethainter	SGuard [46]
Slither	TOD [49]
Slither and Mythril	TIPS [51]
Securify, Slither, Smartcheck	ContractFix [52]
Mythril, Securify, Oyente, Slither	RLRep [56]
Slither	Napoli et al.'s model [58]
HoneyBadger, Osiris, Oyente, Mythril, Slither, and Securify	Ibba et al.'s model [59]
Slither and ANTLR	ACFIX [61]
SOLC_VERIFY	Trusted Deployer [50]
VeriSmart	SmartFix [57]
ML-driven Vulnerability Detection Tools	
LSTM	Feng et al.'s model [42]
eXtreme Gradient Boosting model	SGuard+ [54]
LSTM	SmartRep [55]
Hybrid Vulnerability Detection Tools	
Slither and Random Forest Classifier	Two Timin [60]
Internal Vulnerability Detection Tools	
DeFinery detection tool	DeFinery [26]
HCC detection tool	HCC [48]
GoHigh detection tool	GoHigh [53]
SolSaviour detection tool	SolSaviour [45]
Reparo detection tool	Reparo [47]

Concerning, Figure 4, the pie chart conveys the percentage of vulnerability detection tools that are employed by automated repairing vulnerability frameworks in SCs. The chart reveals that a substantial portion is attributed to traditional analysis tools, while the remaining 29% is divided among internal methods (16%), machine learning-driven methods (3%), and hybrid methods (3%).

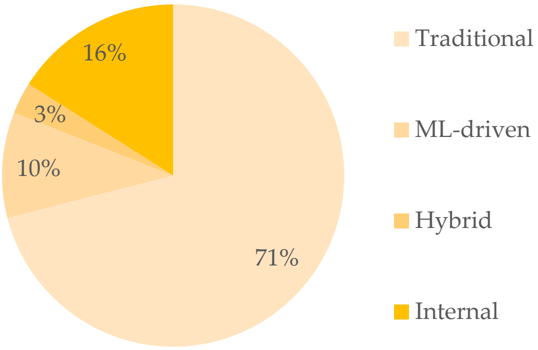

Figure 4. Percentage of vulnerability detection tools employed by automated repairing vulnerability frameworks in SCs.

7. Discussion

This section highlights the challenges and critical findings from our experiments, providing valuable insights into automated SC repair techniques.

Various investigations have been performed to rectify vulnerabilities, drawing upon a diverse set of APR techniques, including search-based, constraint-based, template-based, learning-based, and LLM-based methods. These frameworks take advantage of both bytecode-level and source code-level rewriting methodologies. Strategies at the source code-level provide the benefit of ensuring that patches remain human-readable, thereby preserving all relevant information during the repair process. Nevertheless, the availability of source code is not guaranteed at all times. Fixing bugs is generally simpler through source code modifications than through bytecode-level strategies. Furthermore, bytecode often fails to preserve crucial semantic details.

It seems that the methods currently available are insufficient, as they do not achieve complete automation in detection and correction alongside safety assurances. In addition, several existing tools are plagued by problems, including false-positive localization and high gas expenditure. Furthermore, several of the tools that were utilized are not accessible to the public, thus making it challenging to conduct comparisons and reproduce the research outcomes. Moreover, various identified vulnerabilities have yet to be resolved, and some existing patches do not sufficiently protect against the vulnerabilities they aim to address. These limitations pose severe difficulties for real-world applications.

Techniques for APR based on LLMs have emerged as a leading area of research in contemporary studies. The exclusive use of these methodologies has not yielded successful outcomes in addressing vulnerabilities in SCs, according to the frameworks proposed by Napoli and Gatteschi [58] and Ibba et al. [59]. Nevertheless, the ACFIX [61] and Two Timin [60] paradigms reveal that LLMs can be improved through innovative methodologies. Emphasis needs to be placed on primary concerns, particularly the simplicity of the patch and the costs involved. Moreover, the existing automated SC repair techniques have largely overlooked unknown vulnerabilities.

The focus of off-chain repair tools, including SCRepair, SGuard, SmartShield, and EVMPatch, is to rectify SCs prior to their deployment. In contrast, on-chain repair tools, such as Aroc, are utilized for addressing issues in contracts that have already been deployed. Concerning the execution mechanism of SCs in blockchain, it is essential to take into account two significant aspects: known vulnerabilities and unknown vulnerabilities. When a vulnerability detection tool can identify unknown vulnerabilities, it is reasonable to use off-chain repair tools. However, if it cannot detect these vulnerabilities, on-chain repair tools are a more effective option. Our studies indicate that the majority of current vulnerability detection and repair systems fail to tackle both tasks simultaneously. In other words, most researchers apply the vulnerability detection tool discussed before in their suggested structure. They then attempt to fix the vulnerability by offering a method for repair. A well-structured system is likely to feature the following aspects: a machine learning-oriented detection tool aimed at identifying both known and unknown vulnerabilities, as well as a repair mechanism that integrates off-chain and on-chain methodologies for addressing these vulnerabilities. Moreover, given the developments in LLMs, it is becoming more feasible to apply these methodologies in the context of off-chain repair tools.

8. Conclusions and Future Work

Recently, a notable increase in interest has been observed in the application of template-based APR within SCs. However, LLM-based methodologies have introduced new prospects in this field. This SLR conducted a search query to identify articles published from 2020 up to 21 June 2024. Ultimately, we examined 31 articles published in peer-reviewed scientific research databases, including IEEE, ACM, Springer, Wiley, ScienceDirect, and online archives. The ongoing study did not encompass all existing studies. It excluded non-English articles, editorials, book chapters, surveys, empirical studies, critical articles, technical reports, and master's theses from its scope.

We proposed a taxonomy for automated SC repair under five different perspectives: search-based, constraint-based, template-based, learning-based, and LLM-based approaches. Furthermore, we proposed a taxonomy that categorizes these methods according to their patching levels, explicitly categorizing them into bytecode-based and source code-based types. In addition, we scrutinized the specific types of vulnerability detection tools that each automated repair framework employs.

The principal observations include the following: (1) Research on APR techniques incorporating LLMs has become a leading focus in contemporary studies, with room for improvement through innovative approaches. (2) The primary approach for correcting buggy contracts is rewriting source code, which ensures that the resulting patches are comprehensible to humans. (3) Conventional analysis tools are prevalent in automated SC repair frameworks. (4) Existing automated techniques for SC repair have primarily overlooked the issue of unknown vulnerabilities. (5) It is crucial to prioritize certain desirable features, particularly the simplicity of the patch and the associated costs.

In conclusion, this research provides a foundation for scholars interested in advancing automated SC repair frameworks. We suggest that several challenges are still available and could be worth the attention of researchers. Thus, enhancing the security and reliability of SCs may incentivize a larger cohort of businesses and individuals to integrate blockchain technologies into their operational frameworks.

Scholars are encouraged to embrace hybrid models that effectively leverage LLM-based APR and Self-Paced Learning (SPL) [62]. SPL is a training strategy that involves training on simpler data first, followed by exposure to progressively more complex data. This training technique can be employed during both the vulnerability detection phase and fixing those vulnerabilities. This method has the potential to address the simplicity of the patch and the associated costs.

Author Contributions: Conceptualization, R.K. and V.S.S.; methodology, R.K.; validation, V.S.S.; formal analysis, R.K.; writing—original draft preparation, R.K.; writing—review and editing, R.K. and V.S.S.; visualization, R.K.; supervision, V.S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank the anonymous reviewers and editors for their valuable comments and guidance to make our work more comprehensive.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Qian, P.; Cao, R.; Liu, Z.; Li, W.; Li, M.; Zhang, L.; Xu, Y.; Chen, J.; He, Q. Empirical review of smart contract and defi security: Vulnerability detection and automated repair. *arXiv* **2023**, arXiv:2309.02391.
2. Salzano, F.; Scalabrino, S.; Oliveto, R.; Pareschi, R. Fixing Smart Contract Vulnerabilities: A Comparative Analysis of Literature and Developer's Practices. *arXiv* **2024**, arXiv:2403.07458.
3. Kumar, N.K.; Honnungar, N.V.; Prakash, M.S.; Lohith, J. Vulnerabilities in Smart Contracts: A Detailed Survey of Detection and Mitigation Methodologies. In Proceedings of the 2024 International Conference on Emerging Technologies in Computer Science for Interdisciplinary Applications (ICETCS), Bengaluru, India, 22–23 April 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 1–7.
4. Kiani, R.; Sheng, V.S. Ethereum Smart Contract Vulnerability Detection and Machine Learning-Driven Solutions: A Systematic Literature Review. *Electronics* **2024**, *13*, 2295. [CrossRef]
5. Yu, X.L.; Al-Bataineh, O.; Lo, D.; Roychoudhury, A. Smart contract repair. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2020**, *29*, 1–32. [CrossRef]
6. Wu, C.; Xiong, J.; Xiong, H.; Zhao, Y.; Yi, W. A review on recent progress of smart contract in blockchain. *IEEE Access* **2022**, *10*, 50839–50863. [CrossRef]
7. Wang, M.; Xie, Z.; Wen, X.; Li, J.; Zhou, K. Ethereum smart contract vulnerability detection model based on triplet loss and BiLSTM. *Electronics* **2023**, *12*, 2327. [CrossRef]
8. Fei, J.; Chen, X.; Zhao, X. MSmart: Smart contract vulnerability analysis and improved strategies based on smartcheck. *Appl. Sci.* **2023**, *13*, 1733. [CrossRef]
9. Qian, S.; Ning, H.; He, Y.; Chen, M. Multi-label vulnerability detection of smart contracts based on Bi-LSTM and attention mechanism. *Electronics* **2022**, *11*, 3260. [CrossRef]
10. Sujeetha, R.; Akila, K. Improving Coverage and Vulnerability Detection in Smart Contract Testing Using Self-Adaptive Learning GA. *IETE J. Res.* **2024**, *70*, 1593–1606. [CrossRef]
11. Szabo, N. *Formalizing and Securing Relationships on Public Networks*; First Monday: Canton, TX, USA, 1997.
12. 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, 24–29 October 2016; pp. 254–269.
13. Liu, Y.; He, J.; Li, X.; Chen, J.; Liu, X.; Peng, S.; Cao, H.; Wang, Y. An overview of blockchain smart contract execution mechanism. *J. Ind. Inf. Integr.* **2024**, *41*, 100674. [CrossRef]
14. Ali, I.M.; Abdallah, M.M. On Off-chaining Smart Contract Runtime Protection: A Queuing Model Approach. *IEEE Trans. Parallel Distrib. Syst.* **2024**, *35*, 1345–1359. [CrossRef]
15. Liu, B.; Sun, S.; Szalachowski, P. Smacs: Smart contract access control service. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 29 June–2 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 221–232.
16. Frassetto, T.; Jauernig, P.; Koisser, D.; Kretzler, D.; Schlosser, B.; Faust, S.; Sadeghi, A.-R. POSE: Practical off-chain smart contract execution. *arXiv* **2022**, arXiv:2210.07110.
17. Chen, W.; Yang, Z.; Zhang, J.; Liang, J.; Sun, Q.; Zhou, F. Enhancing Blockchain Performance via On-chain and Off-chain Collaboration. In Proceedings of the International Conference on Service-Oriented Computing, Rome, Italy, 28 November–1 December 2023; Springer: Berlin/Heidelberg, Germany, 2023; pp. 393–408.
18. Ali, I.M.; Lasla, N.; Abdallah, M.M.; Erbad, A. SRP: An efficient runtime protection framework for blockchain-based smart contracts. *J. Netw. Comput. Appl.* **2023**, *216*, 103658. [CrossRef]
19. Reno, S.; Priya, S.H.; Al-Kafi, G.A.; Tasfia, S.; Turna, M.K. A novel approach to optimizing transaction processing rate and space requirement of blockchain via off-chain architecture. *Int. J. Inf. Technol.* **2024**, *16*, 2379–2394. [CrossRef]
20. Xian, D.; Wei, X. ICOE: A Lightweight Group-Consensus-Based Off-Chain Execution Model for Smart Contract-Based Industrial Applications. *IEEE Trans. Ind. Inform.* **2023**, *20*, 1895–1906. [CrossRef]
21. Wang, W.; Song, J.; Xu, G.; Li, Y.; Wang, H.; Su, C. Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Trans. Netw. Sci. Eng.* **2020**, *8*, 1133–1144. [CrossRef]
22. Colin, L.S.H.; Mohan, P.M.; Pan, J.; Keong, P.L.K. An Integrated Smart Contract Vulnerability Detection Tool Using Multi-layer Perceptron on Real-time Solidity Smart Contracts. *IEEE Access* **2024**, *12*, 23549–23567. [CrossRef]
23. Liao, J.-W.; Tsai, T.-T.; He, C.-K.; Tien, C.-W. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. In Proceedings of the 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; IEEE: New York, NY, USA, 2019; pp. 458–465.
24. Momeni, P.; Wang, Y.; Samavi, R. Machine learning model for smart contracts security analysis. In Proceedings of the 2019 17th International Conference on Privacy, Security and Trust (PST), Fredericton, NB, Canada, 26–28 August 2019; IEEE: New York, NY, USA, 2019; pp. 1–6.
25. Hwang, S.; Ryu, S. Gap between theory and practice: An empirical study of security patches in solidity. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–17 July 2020; pp. 542–553.
26. Tolmach, P.; Li, Y.; Lin, S.-W.; Liu, Y.; Li, Z. A survey of smart contract formal specification and verification. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–38. [CrossRef]
27. Garfatta, I.; Klai, K.; Gaaloul, W.; Graiet, M. A survey on formal verification for solidity smart contracts. In Proceedings of the 2021 Australasian Computer Science Week Multiconference, Dunedin, New Zealand, 1–5 February 2021; pp. 1–10.

28. Zhang, Z.; Zhang, B.; Xu, W.; Lin, Z. Demystifying exploitable bugs in smart contracts. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 615–627.
29. Chu, H.; Zhang, P.; Dong, H.; Xiao, Y.; Ji, S.; Li, W. A survey on smart contract vulnerabilities: Data sources, detection and repair. *Inf. Softw. Technol.* **2023**, *159*, 107221. [CrossRef]
30. Wang, Y.; Chen, X.; Huang, Y.; Zhu, H.-N.; Bian, J.; Zheng, Z. An empirical study on real bug fixes from solidity smart contract projects. *J. Syst. Softw.* **2023**, *204*, 111787. [CrossRef]
31. Moher, D.; Liberati, A.; Tetzlaff, J.; Altman, D.G.; PRISMA Group. Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. *Ann. Intern. Med.* **2009**, *151*, 264–269. [CrossRef]
32. Huang, K.; Xu, Z.; Yang, S.; Sun, H.; Li, X.; Yan, Z.; Zhang, Y. A survey on automated program repair techniques. *arXiv* **2023**, arXiv:2303.18184.
33. Hou, Z.; Dong, C.; Shang, Y. HermHD: Enhancing smart contract security based on code obfuscation. In Proceedings of the 2023 11th International Conference on Information Technology: IoT and Smart City, Kyoto Japan, 14–17 December 2023; pp. 96–101.
34. Nassirzadeh, B.; Sun, H.; Banescu, S.; Ganesh, V. Gas gauge: A security analysis tool for smart contract out-of-gas vulnerabilities. In Proceedings of the International Conference on Mathematical Research for Blockchain Economy, Vilamoura, Portugal, 12–14 July 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 143–167.
35. Ren, M.; Ma, F.; Yin, Z.; Fu, Y.; Li, H.; Chang, W.; Jiang, Y. Making smart contract development more secure and easier. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 23–28 August 2021; pp. 1360–1370.
36. Liu, Y.; Mechtaev, S.; Subotić, P.; Roychoudhury, A. Program Repair Guided by Datalog-Defined Static Analysis. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, San Francisco, CA, USA, 3–9 December 2023; pp. 1216–1228.
37. Zhang, Y.; Ma, S.; Li, J.; Li, K.; Nepal, S.; Gu, D. Smartshield: Automatic smart contract protection made easy. In Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 18–21 February 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 23–34.
38. Rodler, M.; Li, W.; Karame, G.O.; Davi, L. {EVMPatch}: Timely and automated patching of ethereum smart contracts. In Proceedings of the 30th Usenix Security Symposium (USENIX Security 21), Vancouver, BC, Canada, 11–12 August 2021; pp. 1289–1306.
39. Jin, H.; Wang, Z.; Wen, M.; Dai, W.; Zhu, Y.; Zou, D. Aroc: An automatic repair framework for on-chain smart contracts. *IEEE Trans. Softw. Eng.* **2021**, *48*, 4611–4629. [CrossRef]
40. Ferreira Torres, C.; Jonker, H.; State, R. Elysium: Context-aware bytecode-level patching to automatically heal vulnerable smart contracts. In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses, Limassol, Cyprus, 26–28 October 2022; pp. 115–128.
41. Guo, S. Automatic Patch Generation System for Smart Contract. In Proceedings of the 2023 IEEE 6th Eurasian Conference on Educational Innovation (ECEI), Singapore, 3–5 February 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 59–62.
42. Feng, Z.; Feng, Y.; He, H.; Zhang, W.; Zhang, Y. A bytecode-based integrated detection and repair method for reentrancy vulnerabilities in smart contracts. *IET Blockchain* **2023**, *4*, 235–251. [CrossRef]
43. Shi, Y.; Zuo, H.; Zhang, Q.; Qin, Z.; Chen, L.; Jiang, X. Automatic Patching of Smart Contract Vulnerabilities Based on Comprehensive Bytecode Rewriting. In Proceedings of the 2023 8th International Conference on Signal and Image Processing (ICSIP), Wuxi, China, 8–10 July 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 763–767.
44. Huang, R.; Shen, Q.; Wang, Y.; Wu, Y.; Wu, Z.; Luo, X.; Ruan, A. ReenRepair: Automatic and semantic equivalent repair of reentrancy in smart contracts. *J. Syst. Softw.* **2024**, *216*, 112107. [CrossRef]
45. Li, Z.; Zhou, Y.; Guo, S.; Xiao, B. Solsaviour: A defending framework for deployed defective smart contracts. In Proceedings of the 37th Annual Computer Security Applications Conference, Virtual Event, 6–10 December 2021; pp. 748–760.
46. Nguyen, T.D.; Pham, L.H.; Sun, J. SGUARD: Towards fixing vulnerable smart contracts automatically. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1215–1229.
47. Thyagarajan, S.A.K.; Bhat, A.; Magri, B.; Tschudi, D.; Kate, A. Reparo: Publicly verifiable layer to repair blockchains. In Proceedings of the International Conference on Financial Cryptography and Data Security, Virtual Event, 1–5 March 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 37–56.
48. Giesen, J.-R.; Andreina, S.; Rodler, M.; Karame, G.O.; Davi, L. Practical mitigation of smart contract bugs. *arXiv* **2022**, arXiv:2203.00364.
49. Beillahi, S.M.; Keilty, E.; Nelaturu, K.; Veneris, A.; Long, F. Automated auditing of price gouging TOD vulnerabilities in smart contracts. In Proceedings of the 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Virtual Event, 2–5 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6.
50. Antonino, P.; Ferreira, J.; Sampaio, A.; Roscoe, A. Specification is law: Safe creation and upgrade of ethereum smart contracts. In Proceedings of the International Conference on Software Engineering and Formal Methods, Berlin, Germany, 26–30 September 2022; Springer: Cham, Switzerland, 2022; pp. 227–243.
51. Chen, Q.; Zhou, T.; Liu, K.; Li, L.; Ge, C.; Liu, Z.; Klein, J.; Bissyandé, T.F. Tips: Towards automating patch suggestion for vulnerable smart contracts. *Autom. Softw. Eng.* **2023**, *30*, 31. [CrossRef]

52. Fang, P. CONTRACTFIX: A Framework for Automatically Fixing Vulnerabilities in Smart Contracts. *arXiv* **2023**, arXiv:2307.08912.
53. Xi, R.; Pattabiraman, K. A large-scale empirical study of low-level function use in Ethereum smart contracts and automated replacement. *Softw. Pract. Exp.* **2023**, *53*, 631–664. [CrossRef]
54. Gao, C.; Yang, W.; Ye, J.; Xue, Y.; Sun, J. sGuard+: Machine learning guided rule-based automated vulnerability repair on smart contracts. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1–55. [CrossRef]
55. Zhou, X.; Chen, Y.; Guo, H.; Chen, X.; Huang, Y. Security code recommendations for smart contract. In Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Macao, China, 21–24 March 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 190–200.
56. Guo, H.; Chen, Y.; Chen, X.; Huang, Y.; Zheng, Z. Smart contract code repair recommendation based on reinforcement learning and multi-metric optimization. *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1–31. [CrossRef]
57. So, S.; Oh, H. Smartfix: Fixing vulnerable smart contracts by accelerating generate-and-verify repair using statistical models. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, San Francisco, CA, USA, 3–9 December 2023; pp. 185–197.
58. Napoli, E.A.; Gatteschi, V. Evaluating chatgpt for smart contracts vulnerability correction. In Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 26–30 June 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1828–1833.
59. Ibba, G.; Ortu, M.; Tonelli, R.; Destefanis, G. Leveraging ChatGPT for Automated Smart Contract Repair: A Preliminary Exploration of GPT-3-Based Approaches. Available at SSRN 4474678. 2023. Available online: https://ssrn.com/abstract=4474678 (accessed on 21 June 2024).
60. Jain, A.; Masud, E.; Han, M.; Dhillon, R.; Rao, S.; Joshi, A.; Cheema, S.; Kumar, S. Two Timin': Repairing Smart Contracts With A Two-Layered Approach. In Proceedings of the 2023 Second International Conference on Informatics (ICI), Noida, India, 23–25 November 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 1–6.
61. Zhang, L.; Li, K.; Sun, K.; Wu, D.; Liu, Y.; Tian, H.; Liu, Y. Acfix: Guiding llms with mined common rbac practices for context-aware repair of access control vulnerabilities in smart contracts. *arXiv* **2024**, arXiv:2403.06838.
62. Kumar, M.; Packer, B.; Koller, D. Self-paced learning for latent variable models. In Proceedings of the Advances in Neural Information Processing Systems (NIPS) 23, Vancouver, BC, Canada, 6–9 December 2010.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.