Titre: Title:	Securing Blockchain Networks and Applications Against DoS Attacks and Smart Contract Vulnerabilities
Auteur: Author:	Fatemeh Erfan
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Erfan, F. (2025). Securing Blockchain Networks and Applications Against DoS Attacks and Smart Contract Vulnerabilities [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/67718/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/67718/
Directeurs de recherche: Advisors:	Martine Bellaïche, & Talal Halabi
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Securing Blockchain Networks and Applications Against DoS Attacks and Smart Contract Vulnerabilities

FATEMEH ERFAN
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Août 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Securing Blockchain Networks and Applications Against DoS Attacks and Smart Contract Vulnerabilities

présentée par **Fatemeh ERFAN**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Alejandro QUINTERO, président
Martine BELLAÏCHE, membre et directrice de recherche
Talal HALABI, membre et codirecteur de recherche
Omar ABDUL WAHAB, membre
Paria SHIRANI, membre externe

DEDICATION

To my father,
who taught me to walk with faith when the road disappeared beneath my feet.

To my sister,
Whose love, a light in the darkness, kept me breathing, kept me believing.

To the memory of my mother,
whose voice still echoes in my dreams, urging me forward with the beauty of her spirit and a quiet strength that supported me in silence. . . .

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisors, Professors Martine Bellaiche and Talal Halabi, for their support, trust, and belief in my potential throughout this academic journey.

Professor Martine Bellaiche has been a steadfast mentor whose support exceeded academic boundaries. Her guidance at every step provided not only critical insight but also reassurance, motivation, and strength during the most trying moments. I am deeply grateful for her generosity, patience, and enduring encouragement, which have been essential to my personal and academic growth.

Professor Talal Halabi, with his profound insight and gentle presence, has been a guiding light on my doctoral path. His thoughtful mentorship, encouragement to explore complex ideas, and ability to challenge conventional thinking have been instrumental in shaping the intellectual depth and clarity of this work. His academic integrity and kindness have left a lasting imprint on me.

This thesis also reflects the boundless love and support I have received from my beloved father, MohammadBagher, my sister, Narges, and my friends. I owe an immeasurable debt of gratitude to my family, who nurtured my curiosity and supported my academic pursuits from the very beginning. Their strength, sacrifices, and encouragement have been the foundation of all I have achieved.

I am especially thankful to Mohammad, whose quiet presence has been a source of strength and patience. His steady belief in me, even in the most difficult times, as well as our shared academic efforts, brought comfort, resilience, and clarity to this journey. His kindness and support have accompanied me throughout, uplifting both my personal and academic path.

Finally, this dissertation is not only the result of my efforts but also a testament to the collective care, wisdom, and compassion of all those who have walked beside me. Throughout this journey, I have truly valued how powerful it is to ask questions, how meaningful and fruitful collaboration can be, and how different perspectives can reshape and enrich our understanding as we seek knowledge.

RÉSUMÉ

Au cours de la dernière décennie, la technologie blockchain a rapidement évolué d'une innovation émergente à un pilier fondamental de l'infrastructure numérique moderne, suscitant un vif intérêt tant dans le monde académique qu'industriel. Son adoption en tant que composante essentielle des systèmes financiers, de l'Internet des objets (IoT), de l'Internet industriel des objets (IIoT), des environnements Metaverse et des plateformes de contrats intelligents s'explique par ses propriétés intrinsèques de transparence, d'immutabilité et de décentralisation. Toutefois, la croissance exponentielle des applications blockchain s'est accompagnée de l'apparition de défis de sécurité sophistiqués, menaçant la stabilité et la fiabilité de ces systèmes.

Parmi les problèmes les plus critiques figurent diverses attaques visant à la fois les couches réseau et applicative. Les systèmes fondés sur la blockchain deviennent de plus en plus vulnérables à des menaces de niveau réseau telles que l'attaque *Eclipse*, le minage égoïste (*selfish mining*) et les attaques Sybil. Les attaquants ont démontré leur capacité à lancer ces attaques pour perturber les services, compromettre l'intégrité des données ou prendre un contrôle disproportionné du réseau. Au niveau applicatif, les vulnérabilités au sein des contrats intelligents—telles que la réentrance, les dépassements et sous-dépassements d'entiers (*overflow/underflow*), ou la dépendance aux horodatages—constituent des failles majeures que les adversaires peuvent exploiter afin de manipuler la logique contractuelle, détourner des fonds ou subvertir le fonctionnement attendu.

Pour répondre à ces problématiques multidimensionnelles, cette thèse propose un ensemble de cadres de sécurité complets, chacun étant adapté à une classe spécifique de menaces et couvrant divers types de blockchains (y compris Bitcoin et Ethereum) ainsi que différents mécanismes de consensus (preuve de travail, preuve d'enjeu, preuve d'autorité). La recherche catégorise systématiquement les stratégies de défense en trois axes principaux : la détection d'intrusion, la mitigation des attaques et la prévention des menaces. Plus précisément, la thèse présente deux nouvelles approches de détection, une technique de mitigation et deux stratégies de prévention.

Au niveau réseau, la première composante vise les attaques *Eclipse* dans les systèmes Metaverse basés sur la blockchain. Le Metaverse—domaine numérique émergent dédié aux activités sociales, éducatives et commerciales—repose fortement sur la blockchain pour la gestion sécurisée des actifs et le contrôle décentralisé. Néanmoins, il demeure vulnérable à des attaques de grande ampleur, notamment les attaques *Eclipse*, susceptibles d'isoler des nœuds

et de faciliter d'autres exploits tels que le *selfish mining* ou la double dépense. Cette étude introduit un cadre de détection d'intrusion s'appuyant sur des algorithmes de détection de communautés, en particulier l'algorithme de Leiden, afin d'identifier avec précision les nœuds malveillants du réseau.

La deuxième composante s'intéresse au minage égoïste, ou *selfish mining*, une menace perturbatrice qui compromet l'intégrité du consensus des blockchains majeures telles que Bitcoin. Cette thèse propose une solution globale couvrant à la fois la détection et la mitigation. Le cadre de détection utilise des ensembles de données simulés du réseau, une sélection avancée de caractéristiques et une classification par forêts aléatoires pour identifier efficacement les comportements malveillants de minage. Pour la mitigation, la thèse développe une taxonomie détaillée des approches basées sur la théorie des jeux, applicables à la sécurité de l'IoT sur blockchain, facilitant ainsi le choix de modèles pertinents pour analyser le comportement des acteurs égoïstes. En particulier, un jeu de la guerre d'usure (war of attrition) est utilisé, où des mécanismes de récompense et de punition orientent les stratégies des mineurs, et la théorie des jeux évolutionnistes permet d'analyser la convergence et la stabilité évolutive de la participation honnête au sein du réseau.

La troisième partie aborde les attaques Sybil dans les environnements IIoT basés sur Ethereum. Les systèmes IIoT, qui relient des processus industriels à grande échelle et des infrastructures critiques, sont particulièrement exposés aux risques posés par la création d'identités fictives visant à compromettre le fonctionnement du réseau. Cette thèse propose un cadre d'apprentissage fédéré décentralisé, où des réseaux de neurones convolutifs sont utilisés de manière collaborative à travers des nœuds distribués pour détecter les comportements Sybil. Par ailleurs, un système de réputation dynamique, piloté par un contract intelligent, est conçu pour empêcher les nœuds Sybil de conserver leur présence dans le réseau, préservant ainsi l'intégrité et l'évolutivité des déploiements IIoT. Afin de soutenir la validation empirique, un simulateur dédié aux scénarios d'attaques Sybil dans des réseaux IIoT basés sur Ethereum est également développé et implémenté.

Au niveau applicatif, la thèse fait progresser la sécurité des contrats intelligents, un domaine essentiel compte tenu de l'immutabilité et de l'autonomie des contrats une fois déployés. La recherche propose deux approches complémentaires : (1) une solution basée sur des modèles de langage à grande échelle (LLM) exploitant l'API GPT pour la détection de vulnérabilités, et (2) une architecture LLM fine-tunée spécifiquement pour l'audit de contrats intelligents. Ces contributions s'appuient sur la création d'ensembles de données exhaustifs et de haute qualité pour la détection de vulnérabilités, ainsi que sur le développement d'une extension en temps réel pour Visual Studio Code (VSCode), fournissant aux développeurs des recom-

mandations de sécurité exploitables directement dans leur environnement de développement.

ABSTRACT

Over the past decade, blockchain technology has rapidly advanced from a novel innovation to a cornerstone of modern digital infrastructure, garnering significant attention from both academia and industry. Its adoption as a core component in financial systems, Internet of Things (IoT), Industrial IoT (IIoT), metaverse environments, and smart contract platforms has been driven by its intrinsic properties of transparency, immutability, and decentralization. However, the exponential growth of blockchain applications has been paralleled by the emergence of sophisticated security challenges that threaten the stability and reliability of these systems.

Among the most critical issues are a range of security attacks targeting both network and application layers. Blockchain-based systems are increasingly vulnerable to network-level threats, such as eclipse, selfish mining, and Sybil attacks. Attackers have demonstrated the ability to launch these attacks to disrupt services, compromise data integrity, or gain disproportionate control over the network. At the application layer, code-level vulnerabilities within smart contracts—such as reentrancy, integer overflow and underflow, and timestamp dependency—constitute critical security weaknesses that adversaries can exploit to manipulate contract logic, drain funds, or subvert intended operations.

To address these multifaceted problems, this thesis proposes several security solutions, each tailored to a specific class of threats and spanning various blockchain types (including Bitcoin and Ethereum) and consensus mechanisms (Proof of Work, Proof of Stake, and Proof of Authority). The research systematically categorizes defense strategies into three primary areas: intrusion detection, attack mitigation, and threat prevention. Specifically, the thesis presents two novel detection approaches, one attack mitigation technique, and two strategies for threat prevention.

At the network level, the first component targets eclipse attacks in blockchain-based metaverse systems. The Metaverse—an emerging digital domain for social, educational, and commercial activities—relies heavily on blockchain for secure asset management and decentralized control. Nevertheless, it remains susceptible to large-scale security attacks, notably eclipse attacks, which can isolate nodes and facilitate further exploits such as selfish mining and double-spending attacks. This study introduces an intrusion detection framework leveraging community detection algorithms, particularly the Leiden algorithm, to accurately identify malicious nodes in the network.

The second component addresses selfish mining, a disruptive threat that undermines the

consensus integrity of prominent blockchains such as Bitcoin. This thesis presents a comprehensive solution encompassing both detection and mitigation strategies. The detection framework leverages simulated network datasets, advanced feature selection, and random forest classification to efficiently and accurately identify malicious mining behaviors. For mitigation, the thesis develops a detailed taxonomy of game-theoretic approaches applicable to blockchain-based IoT security, facilitating the selection of appropriate models for analyzing the behavior of selfish actors. In particular, a war of attrition game is employed, with reward and punishment mechanisms guiding miner strategies, and the evolutionary game theory is then used to analyze the convergence and evolutionary stability of honest participation within the network.

The third part addresses Sybil attacks within Ethereum-based IIoT environments. IIoT systems, which connect large-scale industrial processes and critical infrastructure, are particularly at risk from adversaries creating fake identities to compromise network operations. This thesis presents a decentralized federated learning framework, where convolutional neural networks are employed collaboratively across distributed nodes to detect Sybil behaviors. Additionally, a smart contract-driven dynamic reputation system is designed to prevent Sybil nodes from maintaining network presence, thus preserving the integrity and scalability of IIoT deployments. To support empirical validation, a dedicated simulator for Sybil attack scenarios in Ethereum-based IIoT networks has also been developed and implemented.

At the application layer, the thesis advances the field of smart contract security, a crucial area given the immutability and autonomy of contracts once deployed. The research develops two complementary approaches: (1) a prompt-based large language model (LLM) solution using GPT-API for vulnerability detection, and (2) a fine-tuned LLM architecture tailored to smart contract auditing. These contributions are supported by the creation of comprehensive, high-quality vulnerability datasets and the development of a real-time Visual Studio Code (VSCode) plugin, providing actionable security insights directly to developers.

This thesis introduces a unified and multifaceted framework for safeguarding blockchain-based systems and applications against emerging large-scale threats. Experimental evaluations demonstrate that the proposed solutions achieve superior detection accuracy and efficiency compared to existing methods. By releasing open-source datasets, simulation tools, and auditing frameworks, this work establishes a robust platform for the development of future secure, scalable, and reliable blockchain systems. The insights and tools developed herein are poised to drive ongoing innovation and foster multidisciplinary collaboration across the rapidly evolving landscape of decentralized technologies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Thesis overview

In recent years, the pursuit of decentralization across diverse sectors—including business, education, healthcare, finance, and industry—has witnessed significant momentum [8, 9]. This trend stems from the growing need for trust, transparency, and security in digital ecosystems [10, 11]. Trust has become a central concern for stakeholders seeking to reduce dependency on centralized authorities [12]. Transparency, on the other hand, is a crucial requirement in domains that demand auditability and accountability [9]. Security remains a key challenge, particularly as centralized systems often suffer from vulnerabilities such as single points of failure, which can be exploited by adversaries to disrupt operations or gain unauthorized control [13].

The proliferation of digital technologies has increased the attack surface for malicious actors, who now have more sophisticated tools to manipulate, disrupt, and exploit weaknesses in traditional systems [14]. In response to these concerns, blockchain technology has emerged as a promising solution, offering a distributed, tamper-resistant, and transparent architecture. Initially designed as the foundational infrastructure for cryptocurrencies like Bitcoin, blockchain has evolved into a transformative platform for a wide range of applications, including the Internet of Things (IoT), Industrial IoT (IIoT), Metaverse, financial services, and healthcare.

Despite its core attributes—decentralization, transparency, immutability, fault tolerance, and auditability—blockchain technology is not immune to security threats [15, 16]. As blockchain networks grow in scale and complexity, they become increasingly susceptible to sophisticated attacks that compromise data integrity, disrupt service availability, or exploit protocol-level weaknesses [17]. Of particular concern are Denial-of-Service (DoS) attacks and smart contract vulnerabilities, which can undermine the security and reliability of blockchain-based systems [17]. DoS attacks degrade network performance or render services unavailable, giving attackers an opportunity to manipulate the system or gain control over critical operations, thus putting data privacy and system integrity at risk [18].

Among the various manifestations of DoS attacks in blockchain networks, three forms are particularly prevalent and disruptive: eclipse attacks, selfish mining, and Sybil attacks. Each of these attacks exploits different aspects of the network to degrade performance, compromise consensus, or subvert trust mechanisms. This thesis focuses on these high-impact security

threats, providing an in-depth analysis of their mechanisms and defense strategies.

Eclipse attacks occur when a victim node is isolated from the rest of the network and manipulated through controlled communication channels. In selfish mining, an adversary withholds newly mined blocks to gain a competitive advantage over honest nodes. In Sybil attacks, a malicious entity floods the network with fake identities to dominate consensus or manipulate system behavior.

In addition, the growing adoption of Ethereum and its programmable layer—smart contracts—has introduced a new category of vulnerabilities. Smart contracts, once deployed, are immutable and autonomous, making any flaw or exploit in the code potentially catastrophic [19]. To address these emerging security concerns, this thesis adopts a multi-disciplinary security framework that combines techniques from graph analysis, machine learning (ML), game theory, federated learning (FL), and large language models (LLMs). By integrating these diverse methodologies, we aim to develop comprehensive approaches for detecting, mitigating, and preventing blockchain-based attacks across various domains.

The research particularly emphasizes IoT and IIoT environments, where resource-constrained devices require lightweight yet effective protection mechanisms. It also considers the Metaverse, an evolving digital space where secure identity management and trustworthy data handling are vital for user engagement and asset protection.

To lay the groundwork for this research, the remainder of this chapter introduces the foundational concepts related to blockchain systems, including architecture, consensus algorithms, blockchain classifications, and applications. It also provides an overview of the key attack vectors studied in this thesis, along with the enabling technologies, such as community detection, machine learning, game-theoretic models, and LLMs, that support the proposed security solutions. These concepts form the basis of the architectural and technical contributions presented in subsequent chapters. Following this background, the thesis presents formal problem definitions, articulates the primary research questions, outlines the research objectives, and summarizes the key contributions of this work.

1.2 Blockchain concepts

This section provides the necessary background and foundational knowledge related to the core concepts addressed in this thesis. It introduces key terminologies, technologies, and system architectures that underpin blockchain-based environments, including their applications, security challenges, and supporting methodologies. These definitions are essential for understanding the scope of the proposed work and serve as a conceptual framework for the

analyses, models, and solutions presented in the subsequent chapters.

Blockchain is a decentralized system that allows transactions to be verified by a group of entities [20]. In a blockchain, blocks are created to keep information about the transactions that take place within the network, which may involve the exchange of tokens or any other form of data. To thoroughly understand emerging blockchain-based systems, it is essential to acquire fundamental knowledge about some important examples, blockchain components, types, and important features. Blockchain, a tamper-resistant distributed public ledger, was originally developed for cryptocurrencies such as Bitcoin. The idea of blockchain, first suggested by Nakamoto in 2008 [20], has gained much attention recently as a new peer-to-peer (P2P) technology for decentralized data sharing and distributed computing. The blockchain can resist attacks aimed at taking over control of network systems, thanks to cryptographic technologies and the absence of a centralized control entity.

In 2013, Ethereum was presented as a way to develop blockchain technology by introducing the concept of smart contracts, enabling us to adapt the technology to diverse use cases beyond cryptocurrencies. Blockchain's special characteristics, including auditability, security, transparency, immutability, transactional privacy, integrity, authorization, and fault tolerance [21], make it suitable for a wide range of industries, including IoT, IIoT, Internet of Vehicles (IoV), Metaverse, finance, social services, intelligent transportation, healthcare, and more.

1.2.1 Blockchain components

In this part, key components of blockchain are explained as follows.

Block: In blockchain systems, each transaction is encapsulated within a block. These blocks are linked sequentially to form the blockchain, with the sequence originating from the initial *genesis block* (see Figure 1.1). Each block comprises two principal sections: the header and the body. The body stores transaction data, which is structured as a Merkle tree—an efficient, cryptographically secure data structure in which transactions are hashed in pairs until a single hash value, known as the Merkle root, is obtained [22]. The header contains critical metadata, including the hash of the previous block, a timestamp, a nonce, and the Merkle root. This architecture ensures the immutability and integrity of the blockchain, as any alteration to a block would be immediately detectable through the cryptographic linkage.

Miner: A participant in the blockchain who mines the block is called a miner. After a miner creates a new block, it is added to the blockchain through a process that involves several steps. Initially, miners use a consensus algorithm to solve a cryptographic puzzle. Following

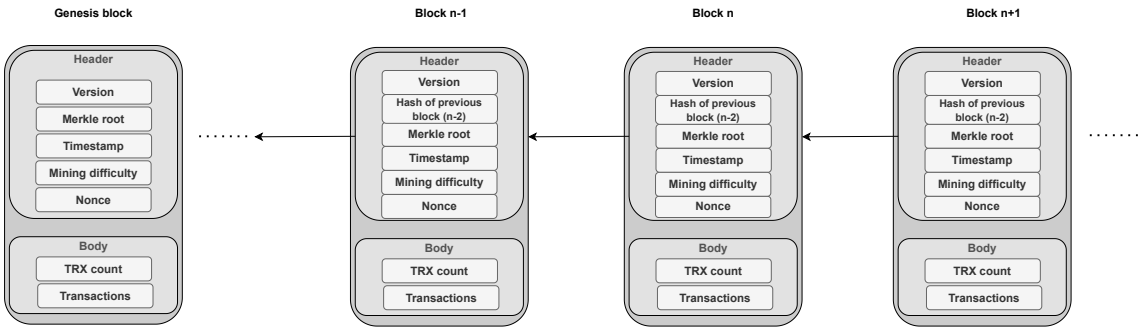


Figure 1.1 The chain of blocks starting from the genesis block.

this, the members validate the new block. It is not mandatory to have a local copy of all transactions to verify a transaction since the verification can be accomplished by using the Merkle tree that is saved within the block. Once the transactions are verified, the miner is rewarded, and the verified transactions are permanently stored in the blockchain.

Consensus algorithm: In general, the validation of the blockchain and the addition of new blocks involve a process known as mining. This process includes verifying transactions within a new block as part of the consensus algorithm. Each transaction has a unique ID, which is the cryptographic hash of the corresponding transaction's data stored within the block.

Proof of Work (PoW): As the earliest consensus algorithm introduced in blockchain technology (notably in Bitcoin), PoW requires participants to solve complex cryptographic puzzles using computational resources. The successful participant, or miner, adds a new block to the blockchain through this resource-intensive process, referred to as *mining*. Miners require powerful computational resources to perform the proof of work and find the nonce. PoW networks have relatively low communication overhead (only needing to broadcast blocks), but achieve low throughput and high latency (confirmation times in the order of minutes) [15,23].

Proof of Stake (PoS): Ethereum officially transitioned to the PoS consensus algorithm in 2022 [24], aiming to significantly reduce energy consumption and provide a resilient platform for novel scaling solutions. Under PoS, participants are referred to as validators rather than miners, and there is no competition to be the first to solve the puzzle by finding the nonce value. Instead, a validator is selected for mining based on their stake in the network, which is determined by their economic share. This selection process is carried out in a pseudo-random manner, with the probability of being chosen proportional to the validator's share [15,25–27]. PoS reduces PoW's computational overhead while keeping network overhead low. It offers higher throughput than PoW and low latency.

Proof of Authority (PoA): [28] It is a recent algorithm that has attracted attention for

its efficiency and fault tolerance, particularly in permissioned blockchain settings. In PoA, a limited set of pre-approved validators take turns proposing new blocks based on a predefined order or time slot mechanism. These validators are trusted entities with known identities, reducing the need for intensive message exchanges. In contrast to alternative consensus algorithms, PoA offers improved performance by streamlining the consensus mechanism and reducing communication overhead. However, the broader implications of these performance improvements, especially regarding availability and consistency within an eventually synchronous network model like the Internet, remain less well understood [15, 29].

Based on existing works [15, 23, 26, 28–33], Table 1.1 classifies the consensus algorithms suitable for IoT networks based on criteria such as accessibility, scalability, computing, network and storage overhead, throughput, and latency. PoA stands out for its low computing and storage overhead, high throughput, and very low latency, making it ideal for private and performance-sensitive IoT environments. In contrast, PoW has very high computing and storage demands with low throughput and high latency, making it less suitable for resource-constrained IoT systems. PoS offers a balanced trade-off with low computing overhead and high scalability, which is suitable for public IoT networks requiring broader accessibility.

Fork: However, even with robust consensus protocols in place, network-level synchronization challenges, particularly the variable time it takes for a newly mined block to reach all nodes, can result in temporary inconsistencies in the blockchain, known as forks.

In practical blockchain networks, latency in block propagation is inevitable, especially given the geographic distribution of miners or validators [34]. When two nodes generate blocks at nearly the same time, and these blocks propagate across the network, some nodes may receive one block first, while others see the other. Each node, by design, builds upon the first block it receives, creating a short-term divergence known as a fork [34]. Such events are considered natural forks and are an inherent aspect of distributed consensus systems. As shown in Figure 1.2, Miner 1 produces block $B1$ while Miner 2 concurrently creates block $B2$. Due to propagation delays, part of the network validates and extends $B1$, while others do the same with $B2$, resulting in parallel branches. This temporary inconsistency is resolved when the network eventually converges on a single chain, typically selecting the branch with the most cumulative work or stake.

Table 1.1 Comparison of consensus algorithms.

Algorithm	**Accessibility**	**Scalability**	**Computing Overhead**	**Network Overhead**	**Storage Overhead**	**Throughput**	**Latency**
PoW	Public	High	Very High	Low	High	Low	High
PoS	Public	High	Low	Low	Medium	Medium	Medium
PoA	Private	High	Low	Low	Low	High	Very Low

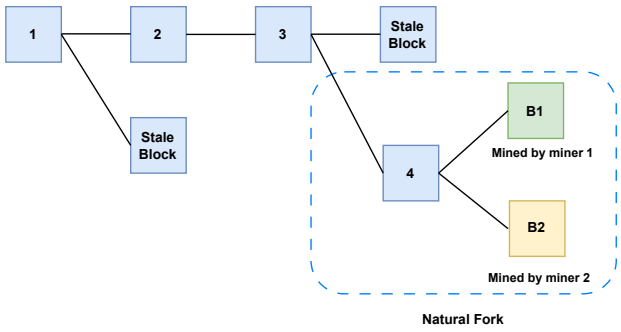


Figure 1.2 Example of a natural fork.

1.2.2 Blockchain types

To better understand how blockchain systems are deployed and governed in various contexts, it is essential to examine the primary categories of blockchain architectures. Blockchain can be classified into different categories based on its utilization and implementation in different systems [15]:

- *Public blockchain*: A permissionless blockchain, also known as a public blockchain, allows anyone to take part in activities such as mining, publishing, verifying new blocks, and accessing the data stored on the blockchain without requiring authorization.
- *Private blockchain*: In a private blockchain, only recognized participants can join the blockchain and retrieve the information shared on the ledger. This form of blockchain is also referred to as a permissioned blockchain.
- *Consortium*: This type of blockchain, also known as a federated blockchain, is a ledger that resembles a private blockchain. It consists of groups of private participants overseen by various organizations. Consequently, multiple organizations with private participants usually form a consortium blockchain.

While blockchain types define the structural and access-related characteristics of a network, their effectiveness and applicability are largely driven by a set of core features that underpin the technology's trust, security, and resilience.

1.2.3 Key features

The characteristics of Blockchain technology, including decentralization, immutability, auditability, fault tolerance, availability, and anonymity, make it an appealing area of study for researchers [25, 35]. Decentralization is one of the most salient characteristics of blockchain.

Turning from a centralized towards a decentralized architecture will protect the IoT network against single points of failure and bottlenecks [36] and enable every device to participate in the consensus protocol. Most participants should verify each transaction in the blockchain, and no central entity monitors the information generated through the network.

- *Decentralization*: Traditional transaction systems rely on a central authority, like a central bank, to validate each transaction generated in the system, which causes cost and performance issues at the central servers. In contrast, blockchain transactions can occur directly between any two peers without authentication from a central agency. This means that blockchain can greatly reduce server costs (including development and operation costs) and address performance problems at the central server [37]. In other words, validating and authorizing data exchanges or transactions in centralized network infrastructures relies on trusted third-party entities, resulting in server maintenance costs and performance bottlenecks. On the other hand, blockchain-based infrastructures allow two nodes to conduct transactions with each other without relying on a central entity to maintain records or perform authorization, thus eliminating the need for centralized trust and associated costs [25]

- *Immutability*: As mentioned before, the Merkle tree technique is used in the blockchain architecture to render the blockchain nearly impossible to tamper with. Thus, any manipulation of a transaction can be revealed. If someone tries to change any of the previous records, they would have to take control of the majority of the nodes within the blockchain network. Otherwise, any modifications made to the blockchain would be quickly noticed [25].

- *Auditability*: Every transaction is stored in the blockchain, and peers hold a copy of the blockchain and can access all time-stamped transaction records. Thus, auditability is another significant feature of the blockchain that enables any participating node to validate the correctness of each transaction that occurred in the blockchain [25, 35].

- *Fault tolerance*: All participants possess a copy of the ledger records. If any fault occurs, it will be revealed through the consensus process, and any data leakage can be cleared using replicas stored by the participants [25].

- *Availability*: Despite distributed ledgers having a high level of data redundancy, with replicated data stored in full nodes, the key benefit of this redundancy is the system's exceptional availability. This characteristic may be precious for applications that cannot tolerate interruptions.

- *Anonymity*: Users' privacy and anonymity can be preserved by using changeable public keys in the blockchain as a method of user identification. Several IoT and Metaverse services and applications, especially those that need confidential identity and privacy protection, find this capability particularly appealing [38].

Building on these foundational features, blockchain technology has been increasingly adopted in various application domains, one of the most prominent being IoT, where security, decentralization, and trust are critical for large-scale deployments.

1.3 Industrial Internet of Things

IoT makes it feasible to connect and communicate with a massive variety of things simultaneously. With IoT, gathering data from the environment using a network of sensors and devices becomes possible. This can lead to improvements in the quality of human life [39]. In the IoT network, the data created, gathered, and processed by network nodes and gateways may contain sensitive and confidential information that security threats can exploit. Traditional cryptographic techniques such as attribute-based encryption [40], identity-based encryption [41], and public key systems [42] have been applied to protect data confidentiality and integrity. However, they are usually expensive with regard to energy consumption, processing, and storage overhead [43]. In addition, they require highly centralized architectures, which raise several deployment and scalability challenges and increase the risk of single points of failure [44].

Blockchain technology has provided a platform for overcoming IoT security concerns and alleviating the inherent bottleneck of fast-growing IoT networks that rely on centralized servers [25]. The security delivered by the blockchain is one of the most crucial aspects that has attracted research on blockchain-enabled IoT systems. Despite mitigating several security threats against IoT systems, including single point of failure, flooding, and jamming [21], other security attacks driven by the blockchain itself could be launched against the IoT platform [45]. As a subset of IoT, IIoT has been introduced to interconnect facilities, systems, and industries by optimizing manufacturing processes, reducing costs, and increasing efficiency in smart enterprises [46]. Blockchain holds significant potential for driving the fourth industrial revolution, transforming all sectors of the economy through its exceptional efficiency. This is largely due to its key features, such as security, immutability, and transparency. Despite these advantages, blockchain networks remain vulnerable to certain types of attacks, including Sybil attacks. In this thesis, we aim to prevent such attacks within IIoT environments.

1.4 Metaverse

Beyond industrial applications, the integration of blockchain into immersive digital systems such as the Metaverse introduces a distinct set of security challenges. Blockchain technology also plays a key role in the development of the Metaverse, a digital space envisioned as the next stage of the Internet. In this immersive environment, users can interact, engage in gaming, attend virtual events, access educational content, and conduct business through their digital representations. The term *metaverse* was first introduced by the science-fiction writer Neal Stephenson in his 1992 novel Snow Crash [47], where it described a 3D virtual world enabling interaction with digital objects and other users. Today, the term broadly encompasses immersive virtual and augmented reality systems, forming a connected digital ecosystem.

The Metaverse integrates various advanced technologies such as blockchain, Virtual Reality (VR), Augmented Reality (AR), and Artificial Intelligence (AI), enabling secure, decentralized management of digital assets [48]. As it collects large volumes of personal data to tailor user experiences, safeguarding this information becomes critical, especially to prevent misuse by unauthorized actors. A decentralized infrastructure is essential to mitigate risks like single points of failure or control by malicious entities. However, coordinating consensus across diverse participants in such a vast environment remains a major technical challenge [47]. Blockchain empowers users by giving them control over their personal data, ensuring enhanced privacy and security.
In a decentralized blockchain-based system for recording transactions across the network, users can manage their digital assets without relying on third parties [20]. This can create a more transparent virtual economy where users can securely exchange their assets and preserve data privacy and security, as blockchain can make it harder for data to be manipulated by unauthorized access. Thus, blockchain could be beneficial in building a secure, transparent, and equitable Metaverse. Nevertheless, blockchain networks are still susceptible to various attacks, such as eclipse attacks. This thesis focuses on the early detection of such threats within IIoT environments.

Although blockchain technology holds significant promise for increasing transparency and granting users enhanced control over their data, particularly in innovative fields such as the Metaverse, it continues to face a spectrum of security challenges. Thoroughly understanding and addressing these vulnerabilities is essential to ensure the reliable and trustworthy deployment of blockchain-based systems.

1.5 Decentralized applications

Decentralized applications (DApps) are software systems that, instead of relying a centralized server, run on a decentralized network, typically a blockchain [49]. Their back-end functionality is implemented as autonomous *smart contracts* that run identically on every validating node, while user interfaces invoke these contracts through middleware such as Web3 libraries [50]. This architecture grants DApps four defining attributes: openly published source code, a native cryptographic token for in-system value transfer, consensus-driven state updates, and fault tolerance derived from full decentralization [51].

Because the code of smart contracts becomes effectively immutable once deployed, even subtle vulnerabilities can propagate irreversible, system-wide losses. Consequently, comprehensive assurance measures, including formal verification, continuous auditing, and real-time on-chain monitoring, are essential to safeguard user assets, maintain composability, and preserve the long-term integrity of the DApp ecosystem [51].

1.6 Attacks on blockchain

As mentioned before, blockchain has drawn tremendous attention over the last years due to its salient features. However, blockchain security has recently been questioned in several attacks, including DoS attacks. A DoS attack is a growing cybersecurity concern where an attacker tries to block legitimate users from using a network, system, or service by flooding it with traffic or resource demands. The particular form of DoS attack varies depending on the target system. Still, they typically involve flooding the system with traffic, taking advantage of security bugs, or causing the system to crash or freeze.

DoS attacks can target devices, networks, and connected services in IoT, IIoT, and Metaverse environments. In these environments, they may disrupt critical operations and degrade the performance of resource-constrained devices, potentially impacting industrial processes and safety systems. In the Metaverse, such attacks could result in widespread disruptions to the virtual world, preventing users from engaging in virtual activities or accessing digital assets.

1.6.1 Eclipse attack

One of the most severe DoS threats to blockchain-based systems is the eclipse attack. This attack enables an adversary to isolate a target node—known as the victim—by monopolizing all of its incoming and outgoing connections. By flooding the victim's routing tables with IP addresses under their control, the attacker effectively filters the node's view of the blockchain

and disconnects it from the rest of the network [52].

Heilman et al. [52] were the first to demonstrate the feasibility of such an attack on the Bitcoin network. They showed that a malicious actor could fill the victim's peer tables with attacker-controlled IPs, allowing only malicious nodes to communicate with the victim. These nodes then send misleading information, lightweight requests, and fraudulent blocks to the victim, replacing honest peers and disrupting normal operations.

As a result, the victim wastes computational resources on invalid data and becomes vulnerable to further attacks. The isolation created by an eclipse attack can serve as a launchpad for more advanced threats, including double-spending attacks, majority (51%) attacks, and block race manipulations.

The implications of such an attack are far-reaching, as an attacker can monopolize all eight of a victim's outbound connections and up to 117 inbound connections, making the isolation nearly complete [53]. This compromises not only the integrity of the victim node but also the stability and trustworthiness of the broader blockchain network.

As illustrated in Figure 1.4, under normal circumstances, a node designated as the victim can freely exchange ledger data with other honest nodes in the Bitcoin network. However, once an eclipse attack is initiated, this communication is disrupted. The victim becomes isolated and can only interact with a set of attacker-controlled nodes, effectively cutting it off from the rest of the network. This compromised view—now entirely shaped by malicious peers—is depicted in the system model shown in Figure 1.3, which represents a Bitcoin network under an ongoing eclipse attack.

In addition to network-level isolation threats such as eclipse attacks, blockchain systems are also exposed to selfish mining, where adversaries exploit protocol incentives to gain disproportionate rewards.

1.6.2 Selfish mining attack

Selfish mining is a strategic and disruptive attack that targets blockchain systems. In this attack, a malicious miner or mining pool intentionally withholds newly mined blocks rather than broadcasting them immediately. By keeping their chain private and selectively releasing blocks at opportune moments, the attacker creates a competitive advantage over honest miners. This behavior leads to a block race between honest and selfish miners, where the latter aims to maintain a lead of at least one block over the public chain [54].

If the selfish miner successfully mines multiple blocks ahead of the honest network, they can override the honest chain upon publication. This results in the orphaning of valid blocks

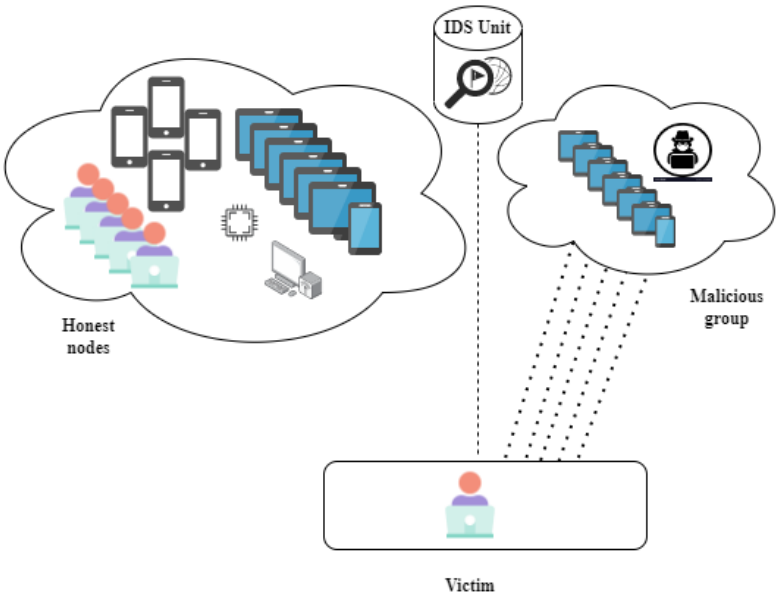


Figure 1.3 System model under the eclipse attack.

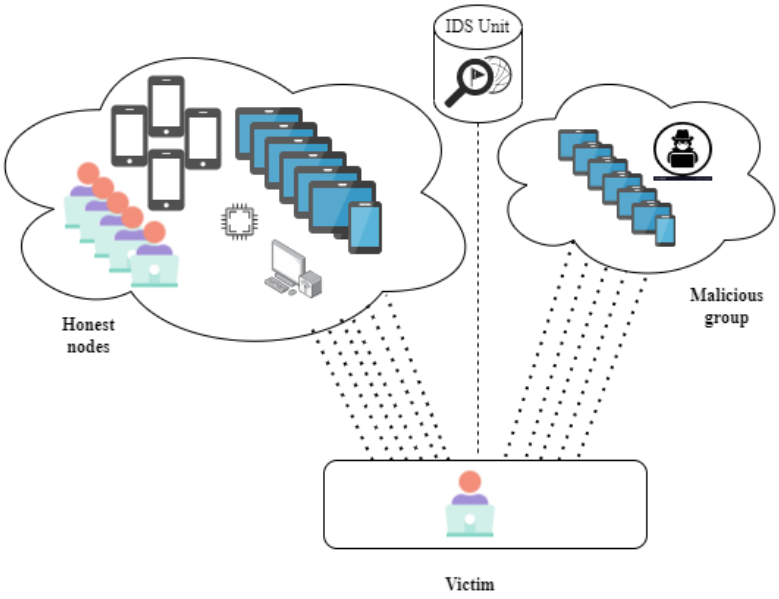


Figure 1.4 System model without the eclipse attack.

mined by honest participants, thereby wasting their computational resources and invalidating the transactions they included. The longer the private chain becomes, the more secure and accepted it appears, increasing the likelihood of being adopted as the main chain once revealed. Consequently, selfish mining not only undermines the fairness of the reward mechanism but also degrades the overall security and decentralization of the network [55].

Figure 1.5 illustrates three representative scenarios of selfish mining. In State 1, the attacker successfully mines blocks A and C before the honest miner produces block B. Upon publishing their chain, the attacker causes block B to be discarded. In State 2, the attacker mines block A, but the honest miner subsequently mines block B and then block C, leading to a tie or loss for the attacker. State 3 shows a case where the honest chain overtakes the attacker's chain, resulting in the attacker losing rewards.

Detecting selfish mining is essential for preserving the integrity of PoW-based blockchain systems. This thesis proposes a machine learning-based detection approach that analyzes network-level indicators. Furthermore, we explore game-theoretic strategies to mitigate such attacks by disincentivizing selfish behavior through appropriate punishment mechanisms.

While selfish mining manipulates consensus mechanisms to target the fairness of block rewards, Sybil attacks undermine network integrity by flooding the system with fake identities, posing a distinct threat to blockchain-enabled environments such as IIoT.

1.6.3 Sybil attack

Sybil attack is another type of DoS attack studied in this thesis, which can be launched on blockchain-enabled systems, including IIoT. In this attack, the attacker creates several fake nodes, called *Sybil nodes*, to gain network control [56]. The adversary introduces a significant number of zero-power entities into the network. These Sybil nodes participate in block propagation but cannot mine new blocks. These fake nodes hinder the propagation of blocks belonging to legitimate users and only forward the attacker's block in the network. As a result, only the attacker's block propagates throughout the network and gets added to the blockchain [57]. In this thesis, we intend to prevent Sybil attacks in blockchain-enabled systems. Figure 1.6 illustrates Sybil attacks in a blockchain-based network.

1.7 Smart contract vulnerabilities

Alongside DoS attacks, blockchain networks are also susceptible to other types of threats, particularly those arising from smart contract vulnerabilities. As mentioned earlier, Ethereum is one of the most widely adopted blockchains, primarily due to the extensive use of smart contracts across various domains.

The concept of smart contracts as digital transactions was initially proposed by Szabo in 1997 [58]. These contracts are autonomous code-based applications that function independently, without requiring third-party involvement. Essentially, they are digital agreements with pre-defined rules enforced through the blockchain. The development of smart contracts

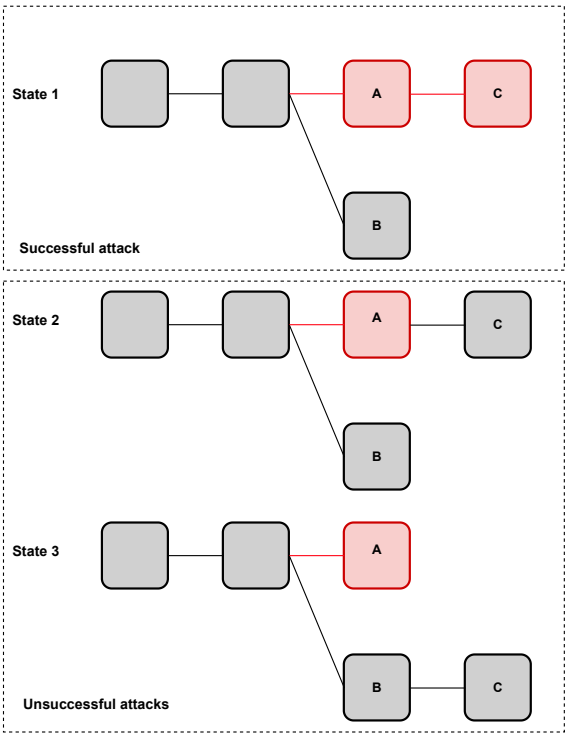


Figure 1.5 Selfish mining attack.

typically involves cooperation among various blockchain users. Once written, the contract is deployed to the blockchain, where it is disseminated across all verification nodes [59]. A smart contract's lifecycle includes four fundamental stages: creation, deployment, execution, and completion.

During the creation stage, the involved parties agree on the contract terms, often with legal input, and software developers translate the agreement into executable code. The deployment phase involves uploading the contract to a blockchain platform, rendering it immutable, and locking associated digital assets. In the execution phase, the contract's functions are triggered automatically when specific conditions are met, and transactions are validated by blockchain nodes. The final phase, completion, results in state updates and asset transfers, effectively unlocking funds for the relevant users. Throughout the contract's operation, all transaction data is logged on the blockchain to preserve integrity and transparency [60].

Despite its robust security framework, blockchain technology is still susceptible to vulnerabilities that can threaten its functionality and trustworthiness. A notable example is the reentrancy exploit on *Cream.Finance*, which led to financial losses exceeding $130 million [61,62]. Critical vulnerabilities impact decentralized finance (DeFi) protocols, their supporting infrastructure, consensus mechanisms, and smart contracts.

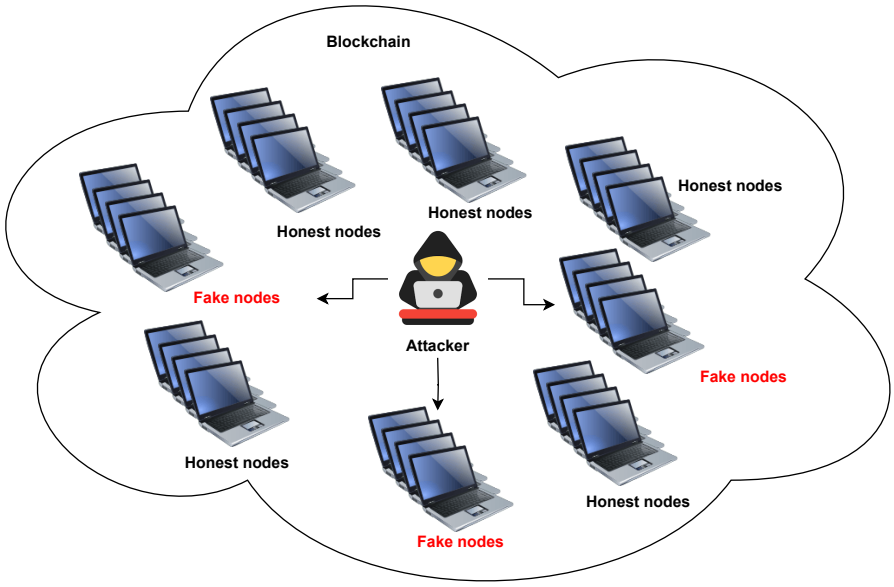


Figure 1.6 Sybil attack.

DeFi protocols are particularly prone to threats like flash loan and oracle manipulation attacks, which can cascade across interconnected protocols [63]. Supporting services—such as wallets, exchanges, oracles, and DApps—can also be compromised, potentially resulting in fund theft, unauthorized data access, and transaction tampering that may affect broader systems.

Attacks on consensus mechanisms exploit algorithmic weaknesses. Examples include double-spending [64], eclipse [52], and selfish mining [54], all of which erode blockchain security and user trust [65]. Various studies have proposed techniques to identify and mitigate these attacks [66–68].

Smart contract flaws remain a critical security concern as attackers can exploit these bugs to disrupt blockchain systems. This study emphasizes major smart contract vulnerabilities that pose the highest risks, as explained below [62, 69].

1.7.1 Integer overflow/underflow (IoU)

Integer overflow and underflow are prevalent bugs in smart contracts [69]. Overflow occurs when a value exceeds the maximum limit of a variable's data type, causing it to wrap around to the minimum value. For instance, incrementing an 8-bit variable at its maximum value

(255) results in an overflow to 0. Conversely, underflow happens when a value goes below the minimum boundary, wrapping around to the maximum (e.g., subtracting 1 from 0 yields 255).

1.7.2 Reentrancy (RE)

A reentrancy vulnerability arises when a smart contract initiates an external call to another contract while its own execution has not yet been finalized. This situation often involves the use of functions such as *call*, *send*, or *transfer* for transferring cryptocurrency to an external address. If the external contract being called is malicious, it may exploit this moment by recursively invoking the original function before the initial execution completes. Such recursive calls can disrupt the expected execution flow and potentially enable an attacker to extract funds from the vulnerable contract repeatedly [69].

For instance, an attacker might deploy a malicious contract and set it up with the address of a target (victim) contract. The attacker triggers a withdrawal after depositing 1 Ether into the target contract. The victim contract verifies the balance and proceeds to send the Ether. However, before the contract can update the internal balance to reflect the withdrawal, the malicious contract re-enters the withdrawal function again. This process can be repeated in a loop, draining the victim's account of its funds until it is either emptied or its computational resources are depleted.

1.7.3 Timestamp dependency (TD)

Within the Ethereum Virtual Machine (EVM), smart contracts function using limited contextual data, primarily depending on the metadata of the block, such as its timestamp. However, this block timestamp is not strictly validated by other nodes, allowing miners to manipulate it arbitrarily. This behavior introduces a vulnerability known as timestamp dependency, which arises when a smart contract utilizes the block's timestamp to perform essential operations. For instance, in a lottery contract, the outcome might be based on a pseudo-random value derived from *block.timestamp*. In such cases, a miner could manipulate the block's timestamp to produce a favorable result, thereby compromising the randomness and fairness of the lottery, ultimately undermining the integrity of the contract. Expert pattern analysis has led to the identification of several patterns associated with timestamp dependency vulnerabilities in smart contracts [62, 69].

1.7.4 Analysis tools

Detection tools for smart contract vulnerabilities fall into two categories: static and dynamic analysis [70]. Static analyzers, such as SmartCheck [71] and Slither [72], review contract source code without execution but often yield high false-positive and false-negative rates. Dynamic analysis tools, including Mythril [73], Oyente [74], Osiris [75], and fuzzing-based tools like ItyFuzz [76], execute contracts to identify runtime vulnerabilities that static methods may miss, though they are more resource-intensive.

Some detectors combine both approaches, but compatibility issues arise, especially with tools like Slither, which require configuration tailored to each contract's Solidity version, hindering scalability for large-scale audits. Due to these accuracy and compatibility limitations, there is a pressing need for more advanced detection methods. This thesis addresses these challenges by proposing an improved LLM-based auditing framework.

Beyond examining smart contract vulnerabilities, this thesis introduces analytical techniques for broader blockchain security challenges. The first technique explored is a graph-based approach using community detection algorithms (CDA), which is particularly effective in identifying structural anomalies such as eclipse attacks within blockchain networks.

1.8 Community detection algorithms

One of the approaches adopted in this thesis to mitigate known attacks is the use of a Community Detection Algorithm. CDA is a powerful tool that can be applied in blockchain environments to tackle various security issues. Due to its wide range of applications, CDA has gained significant interest in fields such as computer science and mathematics. It has been successfully utilized in different domains, including communication, social [77, 78], collaborative [79], and biological networks [80].

Numerous algorithms have been developed to identify community structures across various types of networks. Community detection aims to uncover subsets of nodes that share common characteristics and remain closely connected. For example, in social networking platforms like Facebook or Twitter, CDA can enhance recommender systems by identifying user groups with stronger internal connections [77].

The fundamental objective of community detection is to detect groups or clusters of nodes within a network and classify them into distinct communities. This technique is particularly valuable in the field of network analysis, where understanding the interconnections among nodes can expose underlying structural patterns [81, 82]. Typically, a community is defined as a set of nodes that have denser internal links compared to the links they share with nodes

outside their group [83].

Among the objectives of this thesis is to detect DoS attacks, such as Eclipse attacks, in the blockchain-enabled Metaverse. We propose a novel approach to detecting eclipse attacks against a Bitcoin network based on the concept of community detection.

Although graph-based approaches such as community detection are effective in identifying structural anomalies within blockchain networks, addressing more dynamic and data-driven threats requires the application of machine learning techniques. In this context, machine learning offers robust capabilities for detecting patterns of malicious behavior based on network activity and transactional data.

1.9 Machine learning techniques

Cybersecurity has historically made significant use of data analysis tools. More recently, the spread of cutting-edge machine learning techniques has made it possible to precisely identify cyber attacks and detect risks through active and passive analysis. In particular, machine learning techniques have been used to enable prevention and intrusion detection systems as well as to discover security vulnerabilities in the system [84]. It has proven to be a valuable tool in solving various prediction-related challenges such as intrusion detection, malware identification, anomaly detection, unauthorized identification of IoT devices, DDoS, Sybil, Eclipse, jamming, spoofing, and false data injection attacks. This study aims to use these models to detect security attacks in blockchain systems. This part explains the basic concept of machine learning technology. Two types of machine learning techniques are commonly used in attack detection: supervised and unsupervised.

Machine learning techniques consist of four general steps: data collection, pre-processing and feature extraction, model training and validation, and model testing. As shown in Figure 1.7, data is collected from the network in the first step. Data labeling is needed in supervised learning, while in unsupervised learning, it is not required. In data preprocessing, some operations transform raw data into a suitable format to be used for the next step. Then, the relevant features are extracted either manually or automatically in ML and DL, respectively. Through the model training and validation process, the training set is used to train the model, while the validation set is utilized to obtain the best hyperparameter configuration. Ultimately, the model is evaluated using the testing set, which the model did not see in the previous steps [85, 86].

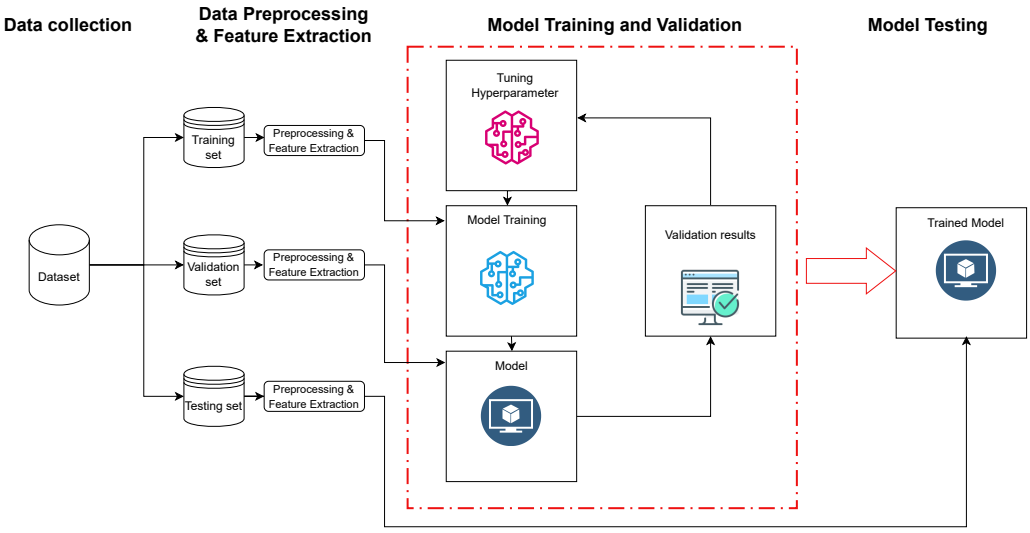


Figure 1.7 General steps in machine learning techniques.

1.9.1 Supervised learning

Supervised learning involves training a machine learning model using labeled data, including normal and malicious behavior, to identify patterns and make predictions on new testing data. In attack detection problems, machine learning techniques can categorize network traffic data into various classes based on security events. A large dataset of labeled network traffic is required. In the training phase, the system's output is optimized to generate a vector of scores that maximizes the likelihood of the desired category. In order to attain this optimal score, an objective function is employed to compute the distance between the predicted and actual labels, enabling the system to update its training parameters that dictate the input-output function of the machine. The learning algorithm computes a gradient vector to adjust the weight vector in the opposite direction of the gradient vector [87].

1.9.2 Unsupervised Learning

An unlabeled dataset is used in unsupervised learning, a machine learning method where the system discovers patterns and structures without direct instruction or supervision. Finding relevant information in the data, such as relationships or clusters, is the objective of unsupervised learning [88]. Unsupervised learning techniques train a model on unlabeled data without human input or direction. It can be utilized to identify malicious behaviors that significantly deviate from typical behavior in the context of blockchain attack detection without needing labeled data. For instance, clustering is one of the well-known unsupervised techniques for attack detection that can be used to find patterns of malicious behavior dif-

fering from normal behavior, which could be signs of security attacks. Anomaly detection can be defined as an unsupervised learning method for attack detection in blockchain-based networks, which entails finding data instances that vary from expected behavior without the need for labeled data. This method is practical for recognizing unknown attacks.

While unsupervised learning is effective in scenarios where labeled data is unavailable, this thesis focuses on a supervised learning approach, leveraging labeled network data to detect selfish mining attacks. Specifically, we employ Principal Component Analysis for feature reduction and Random Forest Classification for accurate and efficient detection.

1.9.3 Principal component analysis

Principal Component Analysis (PCA) is employed in our approach as an effective dimensionality reduction technique to identify and retain the most informative features within the dataset. It transforms the original set of interrelated variables into a smaller number of uncorrelated variables, known as principal components, which capture the most variance in the data. The key advantage of PCA is its ability to reduce the number of features without significant loss of critical information, thereby enhancing the model's efficiency and interpretability [89]. In our study, we utilize PCA to isolate the most relevant network indicators strongly associated with selfish mining attacks in a blockchain network. By filtering out redundant or less impactful features, PCA facilitates a more focused analysis, allowing the Random Forest Classifier to operate on a streamlined feature set while preserving detection accuracy and significantly reducing computational complexity.

1.9.4 Random forest classification

Random forest classification (RFC) is an ensemble learning method that constructs a collection of decision trees during training, where each tree is generated from a randomly selected subset of the training data and features. This randomness ensures diversity among the trees, which enhances the model's robustness and generalization ability. Each tree independently contributes a classification decision, and the final output is determined by aggregating the predictions, typically through majority voting. In our work, this technique is utilized to effectively distinguish between benign and selfish network behaviors. The strength of individual trees and the low correlation between them contribute to the overall accuracy and resilience of the model, making Random Forest particularly suitable for high-dimensional network data where precision and efficiency are critical [90].

Although classification models like RFC can effectively detect malicious behavior, they do not

account for the strategic decision-making of rational adversaries. To model and mitigate such behaviors, especially in attacks like selfish mining, this thesis incorporates game-theoretic approaches that analyze incentives and enforce cooperation in decentralized networks.

1.9.5 Deep learning

Deep Learning (DL), a powerful subset of machine learning, has transformed anomaly detection in cybersecurity by enabling models to automatically learn complex, hierarchical patterns from raw network and behavioral data [91]. Among its most prominent architectures is the Convolutional Neural Network (CNN) [92, 93], which is composed of multiple layers, such as convolutional, pooling, and fully connected layers, each designed to progressively extract and integrate spatial and hierarchical features from structured inputs [94].

Convolutional layers use learnable filters to capture localized patterns, while pooling layers reduce dimensionality and emphasize salient features, though sometimes at the cost of fine spatial detail. Non-linear activation functions like ReLU and sigmoid introduce non-linearity, enabling the modeling of intricate relationships within data, and fully connected layers aggregate the extracted features for final classification. Loss functions quantify prediction errors, guiding the optimization process to enhance accuracy.

For anomaly detection, especially in blockchain-based and industrial IoT networks, communication and behavioral data can be formatted as matrices, allowing CNNs to effectively capture spatial and temporal correlations that distinguish between normal and malicious activities. CNNs' scalability, robust feature extraction, and efficiency make them particularly well-suited for Sybil attack detection in resource-constrained environments, and their capabilities are further amplified when deployed as local models within federated learning frameworks, facilitating privacy-preserving, collaborative training across distributed nodes.

1.9.6 Federated learning

Machine learning can be broadly categorized into three primary paradigms. Localized learning involves training a model exclusively on the data available at a single endpoint. In contrast, centralized learning aggregates data from multiple endpoints and trains the model at a central location, enabling global optimization but raising privacy and data transfer concerns. Federated learning (FL) offers a collaborative approach, wherein models are trained locally at multiple endpoints using their private data, and only the learned model updates, rather than raw data, are communicated to a central aggregator. This process enables privacy-preserving and resource-efficient model development [95].

While FL improves privacy by keeping data decentralized, it traditionally relies on a central server to aggregate model updates, thus introducing a single point of failure (SPF) and potential bottlenecks in coordination and communication. To overcome these challenges, decentralized federated learning (DFL) has emerged as a robust alternative. In DFL, each node independently trains a local model and engages in P2P communication with other nodes, directly exchanging model updates and related metadata, as depicted in Figure 1.8. This decentralized protocol removes the need for a trusted central entity, thereby increasing system resilience, scalability, and fault tolerance [96].

The adoption of DFL is particularly advantageous in distributed, trustless environments such as blockchain-based IoT networks, where centralized coordination is either impractical or undesirable. In the attack defense, DFL offers additional robustness by mitigating the risks associated with server compromise and enabling collaborative anomaly detection across autonomous entities. Nevertheless, DFL also introduces new challenges, including increased communication complexity, trust management among participants, and the need for secure aggregation protocols to prevent adversarial manipulation of model updates.

1.10 Game theory

The integration of blockchain technology into IoT systems brings substantial benefits, including decentralization, data integrity, and improved trust. However, it also introduces new challenges across various domains such as security, consensus mechanisms, communication, energy management, and device heterogeneity. Particularly in security, blockchain can eliminate certain vulnerabilities—like single points of failure—but may also open the door to new classes of attacks such as DoS, eclipse, Sybil, and selfish mining [21, 97].

Game theory has emerged as a powerful framework for analyzing and addressing these adversarial behaviors. Game-theoretic models enable researchers to examine strategic interactions among participating nodes in decentralized systems. These models are especially relevant when nodes may deviate from the protocol to maximize their own utility—for instance, by withholding blocks or launching resource-exhaustion attacks. By modeling such interactions, game theory facilitates the design of incentive-compatible protocols and optimal punishment mechanisms that discourage malicious behavior and promote cooperation.

In blockchain-based IoT environments, game-theoretic approaches also prove useful in resolving non-cooperative optimization problems, such as selecting mining strategies, balancing energy consumption, and distributing computational tasks. Given the constrained nature of IoT devices, these methods help establish efficient resource allocation and reward mechanisms

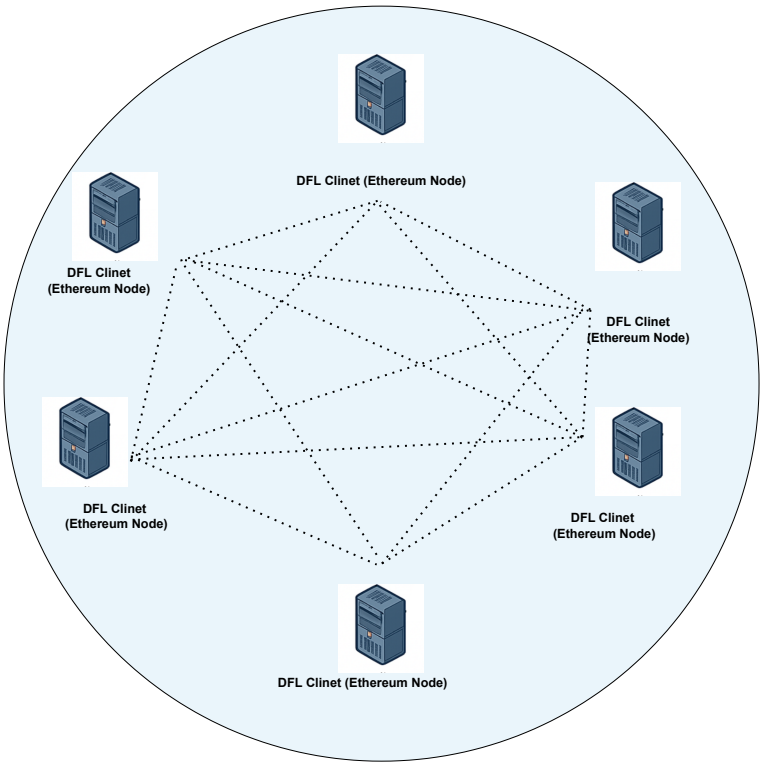


Figure 1.8 Decentralized federated learning in an Ethereum-based network.

under competitive settings [32, 98–100].

1.10.1 Retaliation game

The retaliation game is a class of game-theoretic models in which players deter malicious actions by imposing costs or penalties in response to adversarial behavior. The goal is to discourage harmful strategies by ensuring that attacks are met with consequences, thus maintaining cooperation or system integrity [101–104]. Retaliation strategies are particularly valuable in decentralized systems, where trust is limited and coordination among participants is costly. One prominent subclass of retaliation games is the War of Attrition.

War of attrition game

The War of Attrition, introduced by Maynard Smith [105], represents a more dynamic and resource-driven retaliation strategy. Here, two players compete over a valuable resource by enduring ongoing costs; the contest continues until one concedes. The player who endures longer secures the resource, but at a potentially prohibitive expense. The equilibrium of

this game is characterized by mixed strategies and is both evolutionarily stable (ESS) and subgame perfect (SPE). In the context of blockchain security, the war of attrition framework captures scenarios in which honest nodes must continually bear costs to resist selfish or disruptive miners, ultimately outlasting or deterring them [106]. This approach models a more adaptive and robust retaliation mechanism, particularly when compared to simple mirroring strategies like TFT [107, 108].

1.10.2 Evolutionary game theory

Evolutionary game theory extends traditional models by considering bounded rationality and adaptive behavior over time. Rather than assuming fully rational players, this framework studies how populations evolve toward stable strategies through repeated interactions and imitation. Players adopting successful strategies are more likely to persist and replicate, while less effective strategies are gradually eliminated. In decentralized blockchain systems, this perspective is particularly valuable for modeling long-term adaptation and the natural selection of protocol-compliant behaviors. When applied to security analysis, evolutionary games help assess how strategies like retaliation or cooperation stabilize across large-scale networks and heterogeneous participants [109, 110].

By integrating retaliation-based strategies, evolutionary learning, and dynamic cost modeling, this thesis proposes a game-theoretic framework that captures both the short-term tactical responses to malicious behavior and the long-term adaptation of network participants. This combination allows for robust modeling of selfish mining mitigation strategies through a dynamic, decentralized lens that aligns incentives with honest participation.

While game theory offers insights into strategic behavior in decentralized systems, it is not designed for advanced code analysis or natural language understanding. To address these needs, this thesis adopts Large Language Models (LLMs) for smart contract analysis at the application layer.

1.11 Large language models

In recent years, Large Language Models (LLMs) have been applied across numerous domains, including cybersecurity. The advancement of AI, particularly in the field of natural language processing (NLP), has been significantly influenced by the emergence of LLMs. These models demonstrate remarkable fluency and consistency in both generating and interpreting text. Their effectiveness is largely attributed to the transformer architecture, which serves as a robust framework for sequence modeling and has revolutionized NLP research and applica-

tions [63, 111, 112]. This section introduces the core principles of LLMs and explores their application within the context of blockchain security.

1.11.1 Vulnerability detection standards

In the context of LLM-based vulnerability detection, three principal prompting paradigms are commonly adopted: binary, multi-class, and open-ended prompting [113]. Binary prompting involves producing a simple true or false output indicating the presence or absence of a vulnerability. Multi-class prompting extends this by classifying detected vulnerabilities into specific predefined categories. Open-ended prompting, on the other hand, enables the system to identify and describe any vulnerability, including those that may not belong to a known class.

1.11.2 Enhancement of reasoning process

Several reasoning strategies—such as chain-of-thought (CoT) [114], cumulative reasoning [115], graph-of-thought [116], and tree-of-thought [117]—have been introduced to augment the problem-solving capabilities of LLMs. These methods aim to emulate human-like reasoning by structuring thought processes in a logical and incremental manner. Wei et al. [114] propose CoT prompting, a technique designed to reveal the model's reasoning pathways. This approach emphasizes the generation of natural language explanations for complex arithmetic tasks, along with the use of in-context few-shot learning. Together, these elements enable the model to better address complex reasoning problems by breaking them down into input, intermediate steps, and final outputs. Benchmark assessments have shown that CoT prompting significantly outperforms conventional prompting methods. In addition to enhancing interpretability, this method simplifies intricate tasks into more manageable units. In our methodology, we incorporate the CoT strategy to effectively activate the model's latent reasoning potential.

1.11.3 Generative pre-trained transformers

The year 2023 saw the introduction of several LLMs, notably OpenAI's ChatGPT [118] and Meta AI's LLaMA [119], with ChatGPT amassing over 180 million users worldwide [120]. OpenAI's GPT models have revolutionized natural language processing through their transformer-based architectures and self-attention mechanisms, enabling them to generate highly coherent and context-sensitive text by learning statistical patterns from vast corpora [121]. For instance, GPT-3 comprises 175 billion parameters, endowing the model with an

unparalleled capacity for understanding and generating complex natural language [122].

In this work, we leverage GPT-4o to improve the detection of vulnerabilities in smart contracts, aiming to strengthen the security of decentralized applications on blockchain platforms. The advanced training of these models allows them to generate human-like, contextually relevant outputs, making them a proactive tool for enhancing blockchain security and ensuring safer smart contract deployments.

1.11.4 Parameter-efficient fine-tuning

Fine-tuning LLMs poses substantial computational challenges due to their vast parameter counts, often ranging from millions to tens of billions. Conventional fine-tuning requires updating all model parameters, resulting in significant memory and computational demands that frequently exceed standard research hardware capabilities. To address these limitations, parameter-efficient fine-tuning (PEFT) techniques, such as adapters, bias modules, and low-rank matrices, have been developed to enable task-specific adaptation by introducing a limited number of additional trainable components within select layers of the model while keeping most pretrained parameters frozen [123, 124]. This approach dramatically reduces the resource requirements and often enhances generalization, particularly in scenarios with limited data or computational resources.

Among these methods, Low-Rank Adaptation (LoRA) [125] is especially notable for its effectiveness and simplicity. LoRA introduces trainable low-rank matrices into the attention layers of transformer-based models, allowing only these matrices to be updated during fine-tuning while the original model weights remain unchanged. This not only minimizes memory consumption but also preserves the expressive power of the pretrained model, enabling scalable and efficient fine-tuning even on consumer-grade hardware. Due to their practicality and robust performance, LoRA and related adapter-based approaches have become standard practice for adapting LLMs across diverse applications.

Having introduced the core concepts and technologies underlying this research, the following section details the specific security challenges in blockchain-enabled environments that are addressed in this thesis.

1.12 Problem definition

This thesis addresses four critical security challenges in blockchain-based systems, particularly within emerging and sensitive environments such as the Metaverse, IoT, IIoT, and Ethereum-based smart contract ecosystems. The key concerns are outlined as follows.

1. **Eclipse attacks in blockchain-based systems:**

   Eclipse attacks represent a critical threat to blockchain security, in which adversarial nodes isolate a victim by overwhelming its connections, thereby cutting it off from honest peers. As a result, the victim's transactions and view of the blockchain are manipulated solely by malicious actors, enabling misinformation, delayed propagation, and broader network disruption.

   A fundamental challenge in countering eclipse attacks lies in the fact that the malicious traffic generated during such incidents closely resembles legitimate network behavior. This makes it difficult to distinguish between benign and adversarial activities using conventional traffic analysis or standard feature extraction techniques. Consequently, traditional detection methods often struggle to reliably identify and mitigate these attacks, allowing sophisticated adversaries to evade early detection and maximize the attack's impact.

   Although recent research has explored the use of machine learning and deep learning algorithms for detecting eclipse attacks in blockchain environments, these approaches face significant limitations. Many models struggle to generalize across diverse network conditions and often fail to provide real-time detection capabilities. As a result, attacker nodes frequently go undetected in the initial stages of an attack, increasing the risk of node isolation, ledger manipulation, and escalation to secondary attacks such as selfish mining, double-spending, and network partitioning. This underscores the urgent need for more robust and adaptive detection strategies that can operate in real-time and respond to evolving adversarial tactics in decentralized blockchain systems.

2. **Selfish mining in blockchain-based IoT systems:**

   Integrating blockchain technology within IoT systems is increasingly adopted to enhance security, transparency, and trust among distributed devices. However, this paradigm introduces new attack vectors, with selfish mining emerging as a particularly deceptive threat. In a selfish mining attack, an adversarial miner deliberately withholds newly mined blocks, constructing a private chain rather than broadcasting these blocks to the network immediately. By selectively releasing blocks at opportune moments, the attacker manipulates the consensus process to maximize their own rewards, effectively deceiving honest miners and causing them to waste computational resources on obsolete chains.

   This strategic behavior not only undermines the fairness and efficiency of the blockchain protocol but can also escalate the risk of more severe exploits, such as 51% attacks,

network partitioning, and double-spending. The consequences are particularly acute in IoT environments, where devices often possess limited computational capacity and the timely validation of transactions is critical for operational integrity. Moreover, selfish mining can degrade overall network performance, disrupt service availability, and erode trust among participating nodes.

Given these risks, it is imperative to develop robust and efficient detection mechanisms capable of identifying selfish mining behavior at an early stage. Such solutions are essential for preserving the security, reliability, and sustainability of blockchain-based IoT systems.

While identifying selfish mining attacks is an essential step toward securing blockchain-based systems, detection alone is insufficient to eliminate the threat or address its underlying causes. Even with accurate and timely identification, the economic incentives that motivate rational miners to engage in selfish mining remain unaddressed. As a result, adversaries may continue to exploit vulnerabilities in consensus protocols for personal gain, thereby undermining the overall trust and stability of the blockchain network.

Existing research in blockchain security has predominantly focused on subsequent detection and reporting of selfish mining behavior, while proactive mitigation strategies remain an open problem. As the field advances, there is increasing recognition of the importance of not only identifying such attacks but also developing frameworks that proactively discourage malicious mining activities and reinforce the integrity of the blockchain ecosystem.

Approaches that model interactions among rational actors, such as game-theoretic frameworks, offer promising directions for developing incentive-compatible protocols. However, designing mechanisms that are both practically enforceable and resilient to manipulation remains an open challenge. Therefore, the development of effective mitigation strategies that can adapt to dynamic network conditions and evolving attacker tactics is vital for ensuring the long-term security and fairness of blockchain-based IoT environments.

3. **Sybil attack in blockchain-enabled IIoT systems:**

   As blockchain networks continue to evolve and proliferate across diverse applications, they remain confronted by persistent security challenges, with Sybil attacks constituting one of the most disruptive threats. In a Sybil attack, adversaries generate numerous counterfeit nodes or identities, undermining the integrity of the consensus protocol,

distorting network behavior, and gaining unauthorized influence over critical decision-making processes. By inundating the network with fraudulent identities, attackers are able to manipulate distributed operations, compromise data confidentiality, and erode trust among legitimate participants.

Defending against Sybil attacks is inherently complex, as adversaries frequently mimic the behavioral patterns of honest nodes to evade conventional detection mechanisms. Existing detection and prevention techniques are further constrained by the lack of scalable, privacy-preserving frameworks capable of adapting to the dynamic and heterogeneous nature of contemporary blockchain ecosystems. As the sophistication of Sybil attacks increases, there is a pressing need for comprehensive defense strategies that not only enable accurate identification of Sybil nodes but also protect sensitive data and sustain operational stability without adversely impacting network performance.

This challenge is further accentuated in specialized domains such as IIoT, where blockchain technology is deployed to ensure secure, transparent, and reliable operations. IIoT environments are characterized by highly dynamic communication patterns, resource-constrained devices, and stringent privacy requirements, all of which complicate the detection of Sybil attacks. Lightweight messaging protocols such as MQTT, widely adopted in IIoT, often lack robust built-in security features, rendering them susceptible to identity-based threats. In these contexts, Sybil nodes can disrupt critical industrial processes by transmitting falsified data, overloading communication channels, or subverting consensus mechanisms.

Despite the potential of blockchain to enhance IIoT security, prevailing solutions frequently fall short in addressing Sybil attacks due to limitations in both dataset availability and the scalability of existing approaches. Consequently, there is an evident need for innovative, decentralized, and privacy-preserving detection and prevention frameworks specifically designed for the unique requirements of blockchain-enabled IIoT systems. Bridging this research gap is essential for safeguarding operational continuity, ensuring data privacy, and maintaining the overall trustworthiness of next-generation industrial infrastructures.

Notwithstanding the increasing recognition of Sybil attacks as a critical threat to blockchain-based networks, there remains a pronounced lack of accessible, publicly available simulators capable of realistically modeling, launching, and analyzing these attacks, particularly in IIoT contexts. This deficiency significantly impedes progress in developing and evaluating Sybil prevention mechanisms, as researchers lack reproducible environments for empirical experimentation and benchmarking.

Current research is further limited by the absence of simulation frameworks that can accurately emulate the complex, multi-protocol communication patterns typical of modern blockchain-enabled IIoT networks. As a result, generating realistic, labeled datasets that capture the nuanced behavioral features of Sybil attacks remains a persistent challenge. This limitation not only affects the reliability of detection model evaluation but also hinders fair comparison of competing methodologies and impedes the reproducibility of experimental results.

The lack of an open-source, well-documented Sybil attack simulator thus represents a substantial barrier to empirical research and collaborative innovation within the blockchain security community. The availability of a public simulator would empower researchers to create realistic datasets, test Sybil detection models using a variety of approaches, and facilitate transparent, reproducible, and impactful research aimed at advancing the defense against Sybil attacks. Addressing this gap is critical for the continued evolution and security of blockchain-based systems, especially as they are increasingly adopted in high-stakes industrial and IoT settings.

4. **Smart contract vulnerabilities in Ethereum-based networks:**

   While blockchain technology offers decentralization and immutability, smart contracts deployed on platforms such as Ethereum remain susceptible to a variety of critical security vulnerabilities. In recent years, a series of high-profile attacks exploiting weaknesses in smart contracts have resulted in significant financial losses, with millions of dollars affected across the ecosystem. Notable vulnerabilities, including Timestamp Dependency, Integer Overflow/Underflow, and Reentrancy, have been frequently targeted, highlighting persistent weaknesses in contract logic and execution.

   Despite the proliferation of automated vulnerability detection tools such as Slither, SmartCheck, Oyente, Osiris, and Mythril, these solutions are still inadequate for comprehensive smart contract security. Many of these tools struggle to achieve the level of accuracy required for reliable auditing, often producing high rates of false positives and failing to identify a substantial proportion of vulnerabilities that are ultimately exploited in practice. According to recent studies, as many as 80% of exploited smart contract vulnerabilities remain undetected by existing analysis tools [126]. Inconsistencies further hamper the reliability of these tools in evaluation methodologies and datasets, which can introduce bias and reduce the comparability of results across studies.

   Additionally, the rapid evolution of the Solidity programming language poses another challenge, as numerous traditional analysis tools are built for older versions and lack

compatibility with newer contract features and syntax. This limits their applicability to modern smart contracts and undermines the effectiveness of current vulnerability detection frameworks. As Ethereum-based applications continue to grow in complexity and value, there is a critical need for more accurate, robust, and adaptable vulnerability detection mechanisms that can keep pace with emerging attack vectors and evolving contract standards. Addressing these challenges is essential to maintaining the security, reliability, and availability of Ethereum-based platforms.

A major barrier to advancing research and practice in smart contract vulnerability detection is the absence of large-scale, well-annotated, publicly available datasets. Currently, most studies and tools focus narrowly on identifying vulnerabilities, often overlooking the importance of understanding their broader implications or offering actionable remediation strategies. The lack of high-quality datasets with reliable ground truth labels makes developing, fine-tuning, and rigorously evaluating detection models challenging and contentious.

Many existing datasets suffer from critical shortcomings: they are often incomplete, closed-source, or derived from automated tools prone to high false positive rates [3,127]. Moreover, essential information such as the specific lines of code affected, the vulnerable code segments themselves, and contextual details necessary for robust model training are often missing [5,62,128,129]. These limitations not only impede the development of reliable detection methods but also compromise the validity of experimental evaluations, as the lack of reliable ground truth can lead to biased or irreproducible results.

As the deployment of smart contracts accelerates and the complexity of blockchain applications increases, there is an urgent need to create large-scale, diverse, and structurally rich datasets. These resources are critical for facilitating the practical training, validation, and benchmarking of advanced models, including LLMs. Their availability not only drives innovation but also strengthens the security of Ethereum-based platforms as a whole.

While accurate detection of smart contract vulnerabilities is vital for maintaining the security and integrity of blockchain applications, a significant gap remains in the availability of reliable, real-time detection tools that support developers throughout the contract development lifecycle. Existing solutions are often limited to offline analysis, restricted in functionality, or inaccessible to the wider research and development communities. This lack of readily available and developer-friendly tools impedes proactive security practices, leaving newly deployed contracts susceptible to exploitation.

Moreover, current vulnerability detection tools rarely offer interactive support or inte-

grate seamlessly into the software development workflow. There is a pressing need for advanced tools that can not only identify vulnerabilities in real time as code is written but also provide comprehensive explanations, describe potential security implications and attack scenarios, and deliver actionable guidance for code correction. The absence of such comprehensive and public tools represents a critical barrier to widespread adoption of secure development practices within the Ethereum ecosystem. Addressing this gap is essential for empowering developers to build more secure smart contracts, facilitating early detection of vulnerabilities, and ultimately improving the overall resilience of blockchain-based platforms.

Recent advancements in applying LLMs to smart contract vulnerability detection have shown considerable promise for enhancing the accuracy of automated security analysis. However, there is still a pronounced absence of publicly accessible, fine-tuned models capable of delivering reliable and comprehensive detection of smart contract vulnerabilities. Most existing approaches depend on datasets generated by traditional detection tools, which suffer from high false positive rates, incomplete vulnerability coverage, and inconsistent labeling. As a result, the models built on such datasets often exhibit noisy and insufficiently robust performance.

Moreover, many current LLM-based detection models are either not fine-tuned on diverse, well-structured datasets or are not released for public use, which restricts reproducibility and slows progress in community-driven research. In addition, these models rarely address the remediation aspect of vulnerabilities, further limiting their practical utility for secure smart contract development. The lack of publicly available, thoroughly fine-tuned, and remediation-aware models poses a substantial barrier to advancing secure development practices and the broader adoption of reliable auditing tools. Addressing this gap is essential for fostering reproducible research, promoting innovation, and strengthening the security of Ethereum ecosystems.

1.13 Research questions

These challenges have given rise to the following research questions, which are systematically addressed throughout this thesis:

1. How can eclipse attacks in blockchain-based network environments be detected, given that adversarial traffic closely mimics benign network behavior and existing approaches struggle to generalize across diverse network conditions?

2. How to develop efficient and scalable methods to accurately identify selfish mining behavior in blockchain-based IoT systems, while ensuring timely detection and addressing the resource constraints and security requirements of IoT devices?

3. How can we mitigate selfish mining attacks by proactively discouraging selfish mining behavior and promoting long-term fairness and stability in blockchain-based environments?

4. How to design privacy-preserving frameworks that effectively prevent Sybil attacks in blockchain-enabled IIoT environments, given the inherent limitations of current detection techniques and the complex, heterogeneous architecture of IIoT networks?

5. How to improve the accuracy and robustness of automated smart contract vulnerability detection, particularly for threats such as reentrancy, integer overflow/underflow, and timestamp dependency, in the rapidly evolving Ethereum ecosystem?

In response to these questions, this thesis defines five research objectives to provide secure, scalable, and privacy-preserving solutions for detecting, mitigating, and preventing attacks across blockchain-based systems.

1.14 Research objectives

The research objectives of this thesis are presented to address the security challenges discussed in the previous section.

The main goal is to design, develop, and validate secure and privacy-preserving blockchain-based systems that address critical security challenges—ranging from DoS attacks to smart contract vulnerabilities—while enabling scalable and practical solutions for real-world applications.

1. **To develop a new detection mechanism for blockchain eclipse attacks:** This objective focuses on designing and validating a method that can quickly identify eclipse attacks, which can isolate nodes and disrupt communication within blockchain networks. The goal is to detect these attacks at an early stage to minimize potential damage and ensure continuous, secure network operation.

2. **To propose efficient methods for detecting selfish mining attacks:** This objective involves devising practical and scalable techniques to recognize selfish mining behaviors, where malicious participants manipulate the mining process for personal

gain. The intention is to ensure that mining activities remain fair and that the overall security and stability of the blockchain are not compromised.

3. **To design game-theoretic strategies for mitigating selfish mining attacks:** This objective seeks to develop and evaluate incentive-compatible countermeasures based on game-theoretic modeling, with the aim of discouraging selfish mining behavior by aligning the interests of individual miners with honest participation. By establishing protocols that reduce the profitability of selfish strategies and promote fair resource allocation, this research aims to enhance the overall security, stability, and fairness of blockchain networks.

4. **To propose a privacy-preserving framework for preventing blockchain Sybil attacks:** This objective pursues developing and evaluating a comprehensive system that combines behavioral analysis with reputation tracking to prevent Sybil attacks in decentralized networks. By reducing the influence of malicious entities and reinforcing user confidence in P2P trust, this research aims to enhance the resilience, integrity, and trustworthiness of distributed systems.

5. **To develop automated vulnerability detection tools for smart contracts:** This objective leads efforts to utilize large language models to automatically audit smart contracts for vulnerabilities. By providing intelligent assistance to developers, flagging critical issues prior to deployment, and reducing the risk of costly exploits, this research seeks to strengthen the overall security and reliability of blockchain applications.

   We also strive to create and disseminate comprehensive labeled datasets, simulation environments, open-source real-time plugins, and a fine-tuned LLM. By making these resources available to the research community, this work facilitates reproducibility, fosters innovation, and advances research in blockchain security.

To achieve these research objectives, this thesis presents a series of contributions demonstrating the design, implementation, and evaluation of novel techniques for securing blockchain-based systems across diverse domains.

1.15 Main contributions

This thesis is dedicated to advancing the security of blockchain-based networks, including Bitcoin, Ethereum, blockchain-enabled IoT, IIoT, and Metaverse environments, against a range of security threats and network attacks. The main objective of this research is to strengthen the resilience and trustworthiness of blockchain ecosystems. The proposed methodologies are

systematically applied to diverse blockchain platforms to demonstrate their effectiveness and practical applicability.

This thesis focuses on Eclipse attacks, Selfish Mining attacks, Sybil attacks, as well as enhancing smart contract security to mitigate vulnerabilities in Ethereum networks, one of the most widely used blockchain platforms.

This thesis adopts a comprehensive security approach, addressing detection, mitigation, and prevention across multiple attacks in blockchain networks. While each attack type presents unique challenges, our research follows a structured methodology that strengthens blockchain security at different levels. The thesis presents the following key contributions:

1. **Eclipse attacks detection on blockchain-enabled networks:** We leverage a community detection algorithm to identify and early detect eclipse attacks, which aim to isolate nodes in blockchain networks, particularly in Metaverse environments. This contribution aims to address the first research objective, which is to develop an effective detection mechanism for eclipse attacks.

2. **Detection of selfish mining attacks in blockchain-based networks using machine learning:** We propose a machine learning-based approach utilizing PCA and RFC to detect selfish mining attacks. While demonstrated on a Bitcoin-based network, this method can effectively apply to any blockchain network. This contribution achieves the second research objective.

3. **Mitigating selfish mining attacks in blockchain-based networks using game theory:** After establishing an efficient detection method, we explore mitigation strategies by analyzing game-theoretic approaches. Before proposing a specific defense mechanism, we conduct a comprehensive survey and taxonomy of game-theoretic designs in blockchain-based IoT networks. In this survey, we systematically gather, analyze, and categorize state-of-the-art works applying game theory to address a broad spectrum of challenges, including security, performance, and management issues, in blockchain-enabled IoT systems. By providing a detailed overview of existing models, identifying their applications, and highlighting open research questions, this contribution lays a foundational understanding for future developments in the field. It informs the selection of suitable models for attack prevention and mitigation.

   We adopt a Retaliation Game (a War of Attrition type) as a game-theoretic approach to discourage selfish mining attacks by strategically penalizing malicious actors. This contribution satisfies the third research objective of designing a mitigation mechanism for this attack using a game-theoretic model.

4. **Preventing Sybil attacks in the blockchain-based IIoT networks:** We propose a decentralized federated learning framework to detect Sybil attacks in blockchain-integrated IIoT systems. This approach enhances scalability, preserves data privacy across distributed brokers, and improves detection accuracy through collaborative model training. The framework is integrated with a smart contract to strengthen trust and integrity within the network. This contribution addresses the fourth research objective by modeling and designing a prevention mechanism for Sybil attacks, leveraging federated learning to safeguard data privacy and mitigate the impact of malicious activities on the network.

   In this work, we design a smart contract-enabled reputation scoring system to prevent Sybil nodes from launcging attacks on the network. A dedicated smart contract is deployed to assess and manage the reputation of MQTT brokers. Scores are derived from network behaviors such as peer interactions, messaging anomalies, and connection frequency. Brokers with suspicious activity are identified and can be restricted through a DApp-based interface.

   Moreover, our system confines raw data to local broker nodes, sharing only encrypted or abstracted model updates to safeguard sensitive data. This design ensures collaborative learning and private information protection during federated analysis.

   We also build a simulation tool capable of launching Sybil attacks within an Ethereum-based network, offering a valuable resource for experimentation and validation in security-focused IIoT research.

5. **Securing smart contracts using LLMs:** We address the top three vulnerabilities in Ethereum smart contracts by developing an AI-powered vulnerability detection model using LLMs to prevent exploitable security flaws.

   We advance practical blockchain security by fine-tuning LLMs on a comprehensive, expertly curated vulnerability dataset. Specifically, we fine-tune the open-source LLaMA-3.1 model using parameter-efficient LoRA to enable scalable, memory-efficient training and deployment. This approach offers a transparent, cost-effective alternative to API-based solutions. For comparative analysis, we also fine-tune GPT-4o-mini and systematically evaluate both models against their respective base versions.

   We develop and release a publicly available VSCode plugin that enables security developers to perform real-time analysis of smart contract code. This tool is designed to support both the developer and research communities. This contribution overcomes the research objective of developing a real-time tool with the practical adaptation of security frameworks to detect smart contract vulnerabilities in real time.

We also gather, refine, and significantly expand an existing dataset for smart contract vulnerability detection by improving the quality and granularity of vulnerability annotations. The resulting dataset includes detailed labels, explanations, and code segments for multiple vulnerability types, providing a reliable ground truth for model training and evaluation. By making this dataset publicly available, we enable further research and development in smart contract security and provide the foundation for reproducible and comparable experiments in the field. This contribution directly addresses the final research objective of preparing a comprehensive resource to support and advance the blockchain security research community.

This thesis aims to address the security challenges of blockchain-based systems in IoT, IIoT, and Metaverse environments through a collection of studies focused on eclipse attack detection, Selfish Mining detection and mitigation, Sybil attack prevention, and smart contract vulnerability analysis. The research methods span community detection, machine learning, game theory, federated learning, and the use of LLMs, offering a multi-faceted approach to enhancing blockchain security.

1.16 Thesis structure

Chapter 2 reviews the literature related to the research problems outlined in Section 1.12. This chapter provides a critical analysis of existing studies, identifies their limitations, and highlights the research gaps that this thesis aims to address.

Chapter 3 outlines the research methodology employed in this study. It explains the rationale behind the selected methodological approaches, justifies the use of specific techniques, and presents the alignment between the identified problems and the proposed solutions.

Chapter 4 presents the full text of the article titled *Community Detection Algorithm for Mitigating Eclipse Attacks on Blockchain-Enabled Metaverse*, published in the IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom). The core contribution of this work is the design of an Intrusion Detection System (IDS) to detect eclipse attacks in blockchain-based Metaverse networks using a graph-based method, specifically the Leiden community detection algorithm.

Chapter 5 includes the full text of the article titled *Efficient Detection of Selfish Mining Attacks on Large-Scale Blockchain Networks*, published in the IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C). This work proposes a lightweight detection framework for selfish mining attacks using Random Forest Classification and Principal Component Analysis. The model demonstrates high detection accuracy while significantly reducing computational and temporal complexity compared to deep learning-based approaches.

Chapter 6 provides the full text of the article titled *Retaliation Game for Mitigating Selfish Mining Attacks in Blockchain Networks*, which is to appear in the 13th EAI International Conference on Game Theory for Networks (GameNets). This paper presents a game-theoretic mitigation strategy based on a retaliation game modeled as a war of attrition. The framework incentivizes honest behavior and disincentivizes selfish mining through a combination of punishment and reputation mechanisms.

Chapter 7 presents the full text of the article titled *Sybil Attack Defense in Blockchain-based Industrial IoT Systems using Decentralized Federated Learning*, which has been submitted for review to the Journal of Internet of Things. The main contribution of this paper is a two-phase framework for the detection and prevention of Sybil attacks in blockchain-enabled Industrial IoT networks. The framework incorporates decentralized federated learning and a smart contract-based reputation management system.

Chapter 8 includes the full text of the article titled *Advanced Smart Contract Vulnerability Detection Using Large Language Models*, published in the 2024 IEEE Cyber Security in

Networking Conference (CSNet). This work introduces an LLM-based framework for detecting three critical smart contract vulnerabilities—reentrancy (RE), integer overflow/underflow (IoU), and timestamp dependency (TD). The proposed method outperforms existing tools in terms of recall and F1-score, offering a high degree of accuracy and minimal false positives.

Chapter 9 presents the full text of the article titled *Fine-Tuning AI Models and Implementing a VSCode Plugin for Smart Contract Security*, which has been submitted for review to the Journal of Expert Systems with Applications. This work contributes a publicly available dataset for smart contract vulnerability detection, introduces a fine-tuned LLM for local security analysis, and releases a real-time VSCode plugin to assist developers in secure smart contract development.

Chapter 10 provides a synthesis of how the proposed solutions collectively address the research problems articulated in earlier chapters. It offers a comparative analysis of the effectiveness of these approaches relative to existing methods and critically examines the limitations inherent in the proposed frameworks. In doing so, this chapter offers valuable insights and outlines directions for future research.

Chapter 11 concludes the thesis by summarizing the key problems addressed and the corresponding methodologies developed throughout this research. It also highlights future research directions and potential areas for extending this work.

The Appendix A includes the full text of the article titled *Game-Theoretic Designs for Blockchain-Based IoT: Taxonomy and Research Directions* published in the 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). This survey paper presents a comprehensive review and taxonomy of game-theoretic approaches applied to security, performance, and management challenges in blockchain-based IoT networks. It further outlines future research directions and highlights open challenges for the research community.

CHAPTER 2 LITERATURE REVIEW

This chapter presents a comprehensive literature review aligned with the key research topics addressed in this thesis. The existing works are categorized according to the corresponding research objectives, which include: eclipse attack detection, selfish mining detection and mitigation, Sybil attack detection and prevention, and smart contract vulnerability detection (SCVD) using large language models (LLMs).

2.1 Eclipse attack detection

Blockchain networks such as Bitcoin and Ethereum are susceptible to various network-layer attacks, notably eclipse attacks. In an eclipse attack, an adversary—or a coalition of adversaries—attempts to isolate a target node by monopolizing its peer connections, thereby obstructing its view of the network and disseminating false information. Such attacks can serve as a precursor to more critical exploits. This threat is particularly concerning in metaverse applications, where blockchain plays a pivotal role in securing transactions and ensuring the integrity of digital assets.

In this section, we review existing approaches to eclipse attack detection, evaluating their techniques, deployment environments, and associated strengths and limitations. A common shortcoming among many of these approaches is their lack of generalizability, as they often fail to provide real-time detection or accurately identify malicious nodes. Table 2.1 summarizes a comparison of these methods alongside our proposed approach. We then describe each work in more detail and critically assess their capabilities.

2.1.1 Existing solutions

Initial studies provided the foundational understanding of eclipse attacks. Heilman et al. [52] were the first to demonstrate this vulnerability in the Bitcoin network, showing that only two malicious hosts could successfully isolate a node without the need for additional compromised peers. Marcus [130] extended this insight to Ethereum by proposing low-resource eclipse attacks that exploit the Kademlia-based peer discovery mechanism to isolate nodes with minimal computational effort. While these contributions highlight key vulnerabilities, they do not propose detection mechanisms, thereby limiting their utility in proactive defense strategies.

Machine learning techniques have since been explored to address these limitations. Xu et

al. [131], for instance, applied Random Forest Classification (RFC) to detect eclipse attacks on Ethereum. Although this method demonstrates promising results, its reliance on substantial computational resources may hinder scalability in resource-constrained environments. Moreover, the feature set used is relatively narrow, limiting its ability to capture the full complexity of eclipse attack behaviors.

Alangot et al. [132] introduced a decentralized detection mechanism for Bitcoin networks that combines behavioral analysis with punitive rules based on message transmission patterns. While this approach balances network overhead with detection latency, it struggles to identify malicious nodes, reducing its effectiveness for real-time detection in blockchain-based systems.

Dai et al. [133] employed a deep learning-based approach with a custom feature set to enhance detection capabilities. However, the computational complexity of this method renders it impractical for lightweight or real-time applications.

Similarly, Yves-Christian et al. [53] proposed a framework incorporating misbehavior detection features and a corresponding set of punitive rules to evaluate peer activity across the network. These rules define specific protocol violations, such as oversized payloads or replayed messages, and assign score increments for each, leading to temporary bans once a threshold is reached. Despite its sophistication, this approach also lacks the capacity to identify the attackers themselves accurately.

2.1.2 Discussion

Although substantial progress has been made in detecting eclipse attacks, several critical challenges persist. Foremost among these is the development of accurate, scalable, and real-time detection methods that can effectively identify malicious nodes across diverse and heterogeneous blockchain environments. Considerable computational demands hinder many existing approaches or rely on specific network assumptions, thereby limiting their practical deployment and generalizability. Furthermore, most current techniques emphasize post-attack analysis or reactive mitigation rather than proactive, early-stage detection. This limitation is particularly problematic in rapidly evolving domains such as the metaverse, where blockchain serves as the foundation for the security and integrity of digital assets and transactions. In such contexts, timely and reliable detection of eclipse attacks at the earliest possible stage is essential to prevent subsequent exploits and maintain network trust. Accordingly, there is a critical need for lightweight, adaptive, and robust detection frameworks that can provide prompt and actionable alerts within complex blockchain-based systems.

Table 2.1 Comparison of eclipse attack detection methods.

Study	Platform	Technique	Attacker Identification	Resource Efficiency	Generalization	Environment
Heilman et al. (2015)	Bitcoin	Vulnerability Analysis	No	Yes	No	Theoretical
Marcus (2018)	Ethereum	Vulnerability Analysis	No	Yes	No	Simulation
Xu et al. (2020)	Ethereum	RFC	Yes	No	No	Simulation
Alangot et al. (2021)	Bitcoin	Behavior Analysis + Punishment Rules	No	Yes	Partial	Simulation
Dai et al. (2022)	Bitcoin	Deep Learning	Yes	No	Yes	Simulation
Yves-Christian et al. (2018)	Bitcoin	Misbehavior Detection + Punishment Rules	No	Yes	No	Simulation
Proposed Approach	Bitcoin	CDA	Yes	Yes	Yes	Simulation

2.2 Selfish mining attacks on blockchain-based networks

Blockchain technology, while celebrated for its decentralization and security, remains vulnerable to specific adversarial strategies. One of the most critical threats is the selfish mining attack, wherein malicious miners manipulate block propagation and strategically withhold blocks to disproportionately increase their own rewards. Traditional understanding suggested that control over 51% of the total hashing power was necessary for such attacks; however, Eyal and Sirer [54] challenged this view by demonstrating that attacks can be successful with as little as 25% of the network's hash rate. Subsequent studies further reduced this threshold: Sapirshtein et al. [134] reported a minimum of 23.21%, and Bai [135] noted profitability at 21.48% under certain network conditions.

Defensive strategies against selfish mining in the literature are generally classified into four categories: *detection*, *mitigation*, *prevention*, and *impact analysis*. Across these categories, a range of methodological approaches—including game-theoretic models, machine learning algorithms, and other advanced techniques have been widely adopted to design effective and theoretically grounded countermeasures.

2.2.1 Detection techniques

Detection approaches aim to identify the presence of selfish mining behavior by analyzing blockchain data, network metrics, or miner behavior. Recent research in this area often incorporates statistical analysis, machine learning, and game theory to enhance detection accuracy and adaptability:

- **Machine learning-based detection:** Wang et al. [136] introduced ForkDec, a neural network-based system that distinguishes between honest and selfish forks, achieving 99.03% accuracy. Similarly, Peterson et al. [137] explored block-level indicators using the SimBlock simulator, though their work primarily identified features rather than

deploying a full detection framework.

- **Statistical and lightweight methods:** Chicarino et al. [138] developed a method for monitoring blockchain fork height, flagging deviations as indicators of selfish mining. Saad et al. [66] analyzed block heights and transaction confirmations to classify blocks, while Li et al. [139] used statistical anomalies and address clustering to detect suspicious mining activity.
- **Game-theoretic detection:** Game theory has been used to model miners' incentives, enabling the identification of selfish behaviors based on deviations from equilibrium strategies [97].

2.2.2 Mitigation strategies

Mitigation strategies are designed to reduce the effectiveness or profitability of selfish mining, often by introducing changes to blockchain protocols or incentive mechanisms. Many mitigation approaches incorporate game-theoretic reasoning or leverage machine learning for adaptive defense:

- **Protocol-level modifications:** Heilman et al. [140] proposed incorporating unforgeable timestamps to invalidate delayed blocks. This approach discourages selfish miners by making delayed blocks invalid; however, it imposes the burden of maintaining tamper-proof timestamp records and may not be easily scalable. Zhang and Preneel [141] suggested a fork-resolving policy that prioritizes timely block publication. This method weakens the impact of secretly mined blocks by neutralizing their competitive advantage. While their policy is backward-compatible, it is evaluated under specific assumptions, and its real-world applicability may vary.
- **Resource allocation games:** Luu et al. [142] modeled mining pool dynamics with the Computational Power Splitting (CPS) game, optimizing miner resource allocation to mitigate block withholding attacks. Zhen et al. [143] introduced a Zero Determinant (ZD) strategy, enabling altruistic miners to influence selfish miners' payoffs and encourage cooperation.
- **Adaptive defense via RL:** RL-based mitigation dynamically adjusts protocol parameters or miner strategies in response to detected attacks [144, 145].

2.2.3 Prevention mechanisms

Prevention mechanisms are proactive measures that deter the initiation of selfish mining attacks, often by adjusting miner incentives or introducing strategic disincentives. Game-theoretic modeling is prevalent in this area:

- **Time-based rules:** Lee et al. [146] proposed selecting blocks based on creation time to increase transparency, although this approach requires global time synchronization.
- **Alarming block strategies:** Reno and Sultana [147] proposed an innovative prevention method based on Alarming Blocks (blocks devoid of transactions) issued with notable timestamps to flag selfish mining attempts. This technique effectively suppresses forking without increasing miner overhead. Nevertheless, its applicability is limited to public PoW-based blockchains, and the use of transactionless blocks raises concerns about efficiency and resource utilization.
- **Game-theoretic frameworks:** Eyal [148] modeled pool relationships as non-cooperative games, highlighting that retaliation is only rational for larger pools. Eyal et al. [54] analytically established the minimum hash power threshold for profitable selfish mining. Elliott et al. [149] demonstrated that, under Nash equilibrium, small pools may abstain from retaliation, and Tit-for-Tat (TFT) strategies [150] have been used to promote ongoing honest mining.
- **Comprehensive surveys:** Recent surveys [97, 151], as well as our comprehensive taxonomy of the application of game-theoretic models in blockchain security (which is presented in full text in the Appendix A), summarize a broad range of game-theoretic prevention models for blockchain security.

2.2.4 Impact analysis

Impact analysis seeks to quantify and evaluate the broader consequences of selfish mining attacks on network security, performance, and miner profitability:

- Yang et al. [34] employed Markov models to analyze the influence of block propagation delay and block structure (e.g., uncle-nephew blocks in Ethereum) on attack profitability.
- Kang et al. [152] and Mighan et al. [153] studied how the frequency of forks and block propagation dynamics affect the profitability and impact of selfish mining in both Bitcoin and Ethereum.

Table 2.2 provides a categorized summary of prominent detection, mitigation, prevention, and impact analysis techniques for selfish mining attacks, including their core methodologies and main findings.

2.2.5 Discussion

While the literature offers a rich variety of detection, mitigation, and prevention techniques, often enhanced by game-theoretic or learning-based frameworks, most are developed under specific network assumptions or for particular miner distributions. There is a lack of adaptive, scalable, and context-aware defense mechanisms capable of operating effectively in heterogeneous and dynamic blockchain environments. To address these research gaps, this thesis introduces two complementary approaches for defending against selfish mining attacks:

- **Detection:** A machine learning-based method combining principal component analysis (PCA) and random forest classification (RFC) for accurate, efficient detection of selfish mining attacks, with significant reductions in computational overhead (see Chapter 5).
- **Mitigation:** A novel game-theoretic mitigation framework based on the retaliation game, providing adaptive countermeasures that account for diverse network and miner characteristics (see Chapter 6).

Comprehensive details of the methodologies, experimental results, and theoretical analyses for both detection and mitigation are presented in Chapters 6 and 7, respectively.

2.3 Sybil attack defense in blockchain-based Industrial IoT systems

This section presents existing Sybil defense solutions in blockchain systems and provides an analytical discussion of their shortcomings.

2.4 Existing solutions

Alrubei et al. [154] introduce a hybrid consensus protocol that combines lightweight Proof-of-Work (PoW) and Proof-of-Authority (PoA) through honesty scores reflecting device activity, aiming to select reliable validators in IoT settings. By simulating various attack scenarios, the study evaluates the relative resilience of PoW and PoA in Sybil mitigation. While this approach strengthens validator trustworthiness, it imposes ongoing computational demands on devices with limited resources, potentially reducing energy efficiency and scalability due to the persistent need for score calculation and secure infrastructure.

Table 2.2 Categorization of existing works in selfish mining attacks.

Category	Reference	Methodology	Description
Detection	[136]	NN*	Distinguishes honest vs. selfish forks with high accuracy.
	[138]	Statistical Analysis	Detects selfish mining by monitoring fork height.
	[66]	Structural Analysis	Classifies blocks based on confirmations and heights.
	[139]	Statistical, Clustering	Detects anomalous mining and suspicious clusters.
Mitigation	[140]	Protocol Modification	Invalidates delayed blocks using unforgeable timestamps.
	[141]	Fork-Resolving Policy	Prioritizes timely blocks to counter withheld blocks.
	[142]	CPS*	Models and mitigates BWH via computational power splitting.
	[143]	ZD*	Promotes cooperation via Zero Determinant strategies.
	[144, 145]	RL*	Adaptive protocol/miner strategies in response to attacks.
Prevention	[146]	Time-based Selection	Uses block creation time for fork resolution and transparency.
	[147]	Alarming Blocks	Flags selfish mining using special blocks/timestamps.
	[148]	Prisoner's Dilemma	The pool retaliation and threshold power analysis.
	[54]	MDP*	selfish strategy analysis.
	[149]	Nash Equilibrium	Small pools may abstain from retaliation.
	[150]	Tit-for-Tat	Mutual strategies to sustain honest mining.
	[97, 151]	Survey	Reviews of game-theoretic models for blockchain.
Impact Analysis	[34]	Markov Models	Quantifies profitability and impact of block propagation/structure.
	[152, 153]	Analytical, Simulation	Examines effect of forks and propagation on profitability.

* **Abbreviations:** NN: Neural Network, CSP: Computational Power Splitting game, ZD: Zero-Determinant strategy, RL: Reinforcement Learning, MDP: Markov Decision Process

Kokate et al. [155] propose a trust-enhanced PoA protocol for healthcare IoT systems, where validator election is guided by dynamically updated trust scores to exclude Sybil actors. The method's security, however, depends heavily on the objectivity and accuracy of the trust evaluation process and on the centralized administration of trust data, raising risks of bias, score manipulation, and single points of failure in validator selection.

Sabrina et al. [156] present an Ethereum-based IoT data-sharing scheme that leverages fuzzy extractors to bind device identities, ensuring that only authenticated nodes may join the blockchain and reducing Sybil exposure. The system's effectiveness presumes the existence

of a robust, scalable identity infrastructure, which complicates key management and may present barriers for deployment in diverse IoT environments.

Wang and Tan [157] develop a Practical Byzantine Fault Tolerance (PBFT) variant augmented with reputation mechanisms, granting more influence to nodes with verifiable honest behavior, thereby hindering Sybil penetration in consensus processes. The reliance on a single, high-reputation primary node introduces a centralization risk: if compromised, the network's security and liveness are jeopardized. Additionally, the requirement for persistent behavioral monitoring and message auditing raises communication and storage costs, particularly in expansive networks.

Venkatesan and Rahayu [158] integrate diverse consensus mechanisms—PoW, PoS, DPoS, PBFT, and PoCASBFT—with machine learning models to detect and thwart Sybil and other network attacks. While this layered approach increases defensive robustness, it brings heightened complexity and resource overhead, especially from latency-prone protocols like PBFT and DPoS. Centralization risks and scalability constraints present deployment challenges in IIoT networks, where reliability and efficiency are paramount.

Dewanta et al. [159] design a lightweight cryptographic protocol tailored for MQTT-based IoT systems, incorporating timestamp-based freshness checks, hash-derived session keys, and symmetric payload encryption to combat Sybil attacks. The method achieves confidentiality, integrity, and replay protection with low resource consumption, yet its security depends critically on the MQTT broker's integrity and secure private key management. Assumptions of secure parameter pre-distribution and the lack of a scalable key exchange limit its use in large, dynamic IoT deployments.

Eisenbarth et al. [160] present a Sybil prevention scheme for Ethereum networks, employing centralized monitoring to detect malicious nodes and disseminate alerts via smart contracts. Although effective for detection, this architecture suffers from high gas costs, a single point of failure in the monitoring node, and lacks decentralized enforcement. The necessity for periodic crawling also restricts real-time responsiveness.

Mishra et al. [161] propose modeling attacker behaviors using Markov chains, applying K-means clustering to optimize defensive resource placement, and leveraging identity replacement strategies to enhance stealth detection. Despite its rigorous analytical framework, the approach remains untested in real-world IoT scenarios and relies on assumptions about attacker knowledge (e.g., node energy, proximity) that may not hold in practice.

Zhao et al. [162] propose a reputation framework for IIoT environments where only authenticated participants, verified via taxed-purchase tokens and cryptographic checks, may influ-

ence network trust scores, thus deterring Sybil attacks. This model depends on a trusted tax authority for transaction verification; in its absence, the scheme's reliability is undermined. Moreover, as network scale and transaction rates increase, blockchain-related performance bottlenecks may impede real-time operation and broader adoption.

Prathiba et al. [163] introduce a federated learning security framework for IIoT by integrating NFT-based identity management and digital twins for model verification. NFTs ensure unique device participation, while digital twins serve as static references for aggregation integrity. Despite enhanced security, the system incurs significant blockchain, computation, and storage costs, and is anchored on a centralized setup phase, which introduces a single point of failure. Frequent on-chain transactions and cryptographic verifications also risk reduced throughput in high-volume IIoT deployments.

Table 2.3 provides a comparative overview of recent Sybil attack defense mechanisms in blockchain-based networks, summarizing their core approaches, main techniques, advantages, and limitations.

2.4.1 Discussion

The literature demonstrates significant progress in Sybil attack defense for blockchain-based IIoT networks. Various strategies, ranging from hybrid consensus protocols and trust management to cryptographic schemes and federated learning, address different dimensions of the Sybil problem. Notably, many recent works aim to enhance validator selection, strengthen device authentication, or improve reputation management through innovative protocol design and machine learning integration.

Despite these advancements, several recurring challenges persist. Many solutions introduce additional computational or communication overhead, rely on centralized components that may present single points of failure, or depend on trusted third parties for security guarantees. Scalability, real-time responsiveness, and deployment in resource-constrained or heterogeneous environments also remain open concerns.

Furthermore, the effectiveness of some frameworks hinges on strong assumptions regarding infrastructure, key management, or attacker knowledge, which may limit their practical applicability. As industrial IoT ecosystems continue to evolve, a critical need remains for scalable, decentralized, and adaptive Sybil defense mechanisms that balance security, efficiency, and practicality.

Having established the current state of Sybil attack defense in blockchain-based IoT environ-

ments, the next section reviews advances in smart contract vulnerability detection.

2.5 Smart contract vulnerability detection using large language models

Recent advances in large language models (LLMs) have significantly accelerated research in the automated detection of vulnerabilities within smart contracts, offering new paradigms for both prompt-based and fine-tuned approaches.

2.5.1 Prompt-based and API-driven LLM auditing

Hu et al. [113] introduce GPTLens, a two-stage framework in which an LLM operates first as an auditor and subsequently as a critic, aiming to enhance vulnerability detection in smart contracts. This architecture has shown promise in reducing false positives; however, the initial stage often produces a high volume of potential vulnerabilities, posing challenges for downstream validation and potentially overwhelming human analysts.

Liu et al. [164] present PropertyGPT, a method that leverages LLMs for vulnerability identification and is evaluated on thirteen smart contracts from the *Smart-Bug Curated* dataset [128]. PropertyGPT demonstrates improved performance over GPTScan [165] by identifying more true vulnerabilities, but remains constrained to binary classification and a limited set of vulnerability types. GPTScan [165], which employs GPT-3.5-turbo in conjunction with static analysis, is similarly limited by the precision of static analysis, especially for contracts of greater complexity or size.

Zaazaa et al. [166] employ LLMs, notably ChatGPT with in-context learning, to streamline vulnerability rule integration and detection, reporting exact-match accuracies exceeding 90% with sufficient training data.

Ding et al. [167] introduce SmartGuard, a framework for vulnerability detection that leverages large language models (LLMs) alongside the SolidiFi dataset and chain-of-thought (CoT) prompting. Unlike approaches that employ expert-driven refinement, SmartGuard processes the raw dataset without thorough preprocessing or expert oversight. The prompting strategy is based on semantic retrieval using CodeBERT rather than explicit expert cues. Additionally, the evaluation conducted by SmartGuard does not incorporate expert assessment and overlooks critical issues such as potential dataset leakage and operational cost-effectiveness.

Table 2.3 Comparison of most recent Sybil attack mitigation techniques in blockchain-based IIoT/IoT.

Reference	Domain	Consensus / Mechanism	Technique	Defense Type	Limitations
Mishra et al. (2018) [161]	IoT	Markov Chain + K-Means	Analytical model of Sybil behavior and stealth strategy	Prevention	Not implemented; unrealistic attacker assumptions
Alrubei et al. (2022) [154]	IoT	HDPoA* (PoW + PoA)	Lightweight PoW for honesty score, used in PoA selection	Prevention	Additional computation and complexity in score management
Eisenbarth et al. (2022) [160]	Ethereum	Centralized Crawler + Smart Contract	Periodic detection via crawler, reporting via smart contract	Detection	Centralized monitoring; gas cost; limited real-time response
Sabrina et al. (2022) [156]	IoT	Permissioned BC	Device identity management using fuzzy extractors	Prevention	Requires pre-established identity infrastructure
Wang & Tan (2023) [157]	IoT / BC	PBFT* with Reputation	Reputation scoring to weight PBFT votes	Detection	SPF; communication overhead
Dewanta et al. (2023) [159]	IoT	Lightweight Crypto + Timestamps	Symmetric encryption + timestamp verification for message integrity	Prevention	SPF at broker; weak key distribution assumptions
Venkatesan et al. (2024) [158]	IIoT	Hybrid + ML	Hybrid consensus with ML-based Sybil detection	Detection	complex; high latency; SPF in some components
Prathiba et al. (2024) [163]	IIoT	NFTs + DTs + Smart Contract	Unique NFT identities + DTs validation for model integrity	Prevention	High computation/storage costs; static DTs and centralized init
Zhao et al. (2025) [162]	IIoT / Retail	Reputation System + Tokenization	Purchase-tax-linked tokens for reputation scores	Prevention	Relies on tax authority; adds latency and scalability limits
Kokate et al. (2025) [155]	IoT / HCIoT*	Trust-weighted PoA	Trust score modifies validator power	Prevention	Trust bias risk; SPF

Abbreviation: HDPoA: Honesty-based distributed proof of authority, HCIoT: health-care IoT, PBFT: practical byzantine fault tolerance

2.5.2 Fine-tuned LLMs and domain adaptation

The application of fine-tuned large language models to vulnerability detection has produced notable results. Ince et al. [168] fine-tune Meta's Code Llama and Instruct models (34B parameters each) for this task. Despite their advanced modeling, the authors report that detection outcomes are inconsistent across vulnerability types, in part due to the modest accuracy of supporting tools. The resource demands of such large models (A100 80GB GPUs) further constrain accessibility and practical deployment.

Ma et al. [129] introduce iAudit, a two-stage framework employing majority voting and agent feedback to mitigate hallucinations in LLM-based auditing. Nevertheless, iAudit's reliability is hampered by low output consistency and a small, narrowly scoped dataset (263 contracts, median 35 lines), raising concerns regarding generalizability. Wei et al. [6] present *LLM-SmartAudit*, a collaborative multi-agent system for vulnerability detection. While this approach leverages agent specialization, it lacks evaluation on established benchmarks, does not offer line-level vulnerability localization, and does not generate explicit remediation guidance, which may limit its utility for developers.

Yu et al. [169] propose Smart-LLaMA, which enhances vulnerability detection and explanation quality through a two-stage domain adaptation and explanation-guided fine-tuning process. While the approach produces more detailed reports and explanations, its reliance

on an 8B parameter model entails substantial computational requirements, presenting barriers to efficient real-time or on-device auditing. However, the model is reported to generate repetitive explanations after supervised fine-tuning, indicating a tendency toward output redundancy and limited diversity in its responses.

2.5.3 Benchmarking datasets and evaluation

Ghaleb et al. [170] propose SolidiFI, a robust framework for benchmarking smart contract vulnerability detection, employing systematic bug injection into authentic contracts from Etherscan[1]. Du et al. [5] assess GPT-4's vulnerability detection performance on the SolidiFI dataset, ultimately finding limited effectiveness. Their study highlights the importance of sophisticated prompt engineering, as insufficient design can lead to results inferior to random classification. This underscores the need for more robust evaluation protocols and datasets in this field.

2.5.4 Discussion

Current research demonstrates that both prompt-based and fine-tuned LLMs have advanced the detection of smart contract vulnerabilities, but significant challenges remain. Many existing approaches rely on small datasets, which limits model generalizability to real-world contracts. Some existing works focus on only a subset of vulnerability types or use synthetic benchmarks without additional validation in live deployments, which may not fully capture real-world complexity. In addition, several methods do not provide line-level localization or concrete remediation steps, reducing their immediate utility for developers who need precise and actionable fixes. In terms of performance, some approaches still require resource-heavy models, creating barriers to real-time or on-device auditing.

Finally, persistent issues such as model hallucinations (e.g., repetitive or fabricated explanations after fine-tuning), dataset leakage risks, and insufficient evaluation rigor remain open challenges. Future research will benefit from more comprehensive datasets, efficient model architectures, and advanced evaluation methodologies to improve the reliability, scalability, and applicability of LLM-driven smart contract security.

2.6 Conclusion

This chapter provided a comprehensive and structured review of the existing literature across the core research topics addressed in this thesis. We began by examining eclipse attack detec-

[1] `https://etherscan.io`

Table 2.4 Comparison of recent LLM-based smart contract vulnerability detection frameworks.

Reference	LLM Model	Framework	Technique	Limitations
Ghaleb et al. (2020) [170]	N/A	SolidiFI	Benchmarking and dataset generation	Not a detection system; focuses on benchmarking
Hu et al. (2023) [113]	GPT-based (API)	GPTLens (Auditor + Critic)	Two-stage LLM auditing; reduces false positives	High volume of candidates in first stage; validation bottleneck
Liu et al. (2024) [164]	GPT-based (API)	PropertyGPT	LLM prompt-based binary detection	Small sample size; only binary detection; limited vuln.* types
Sun et al. (2024) [165]	GPT-3.5-turbo	GPTScan	LLM + static analysis for detection	Limited by static analysis precision; struggles on large/complex contracts
Zaazaa et al. (2024) [166]	ChatGPT	SmartLLMSentry	Rule-based detection with in-context LLM	Dependent on sufficient data; not evaluated for localization
Ince et al. (2024) [168]	Code Llama, Instruct (34B)	Fine-tuned LLM	Fine-tuned LLMs for vulnerability detection	High hardware cost (A100 GPUs); inconsistent detection
Ma et al. (2024) [129]	GPT-3.5	iAudit	Two-stage, majority voting, agent feedback to mitigate hallucination	Small, short contracts; low output consistency; generalizability concerns
Wei et al. (2024) [6]	GPT-based (API)	LLM-SmartAudit	Multi-agent conversational detection	No established benchmark; lacks localization and fix suggestions
Yu et al. (2024) [169]	LLaMA-3.1 (8B)	Smart-LLaMA	Domain-adapted pretraining + explanation-guided fine-tuning	Generate repetitive explanations after FT*
Du et al. (2024) [5]	GPT-4 (API)	Prompt-based GPT-4	Prompt-based GPT-4 on benchmark	Weak prompt engineering; underperforms random classification
Ding et al. (2025) [167]	LLM + CodeBERT	SmartGuard	CoT*, CodeBERT semantic retrieval	Unrefined dataset; lacks expert validation; no cost analysis; no explicit expert guidance; less robust prompt design

Abbreviations: FT: fine-tuning, vuln.: vulnerability, CoT: chain of thought

tion methods in blockchain-based networks, highlighting recent advances in detection techniques. Next, we offered an in-depth analysis and categorization of the literature on selfish mining attacks, systematically organizing prior work into four principal domains: detection, mitigation, prevention, and impact analysis.

We then reviewed recent developments in Sybil attack defense within the context of IIoT and

blockchain-enabled environments, emphasizing the evolution of decentralized, reputation-based solutions. Following this, the chapter explored the rapidly advancing field of smart contract vulnerability detection using large language models, distinguishing between prompt-based and fine-tuned approaches and analyzing benchmarking practices.

Despite significant progress in each area, the literature reveals ongoing challenges in achieving accurate, scalable, and real-time detection and prevention across heterogeneous blockchain environments. The need for adaptive and efficient mechanisms remains critical. These gaps underscore the importance of advancing integrated approaches to detection, mitigation, and prevention to enhance the security and resilience of modern blockchain-based networks.

CHAPTER 3 RESEARCH METHODOLOGY

This chapter outlines the rationale underlying the methodological choices made throughout this thesis. Given the heterogeneity of applications and architectures in blockchain-based systems, our research adopts a multi-domain perspective and develops tailored strategies to address the security challenges of each domain.

Accordingly, we establish a comprehensive security framework by systematically integrating 1) graph-based community detection for eclipse attack detection, 2) machine learning techniques for identifying selfish mining, 3) game-theoretic modeling for the mitigation of selfish mining attacks, 4) federated learning for scalable and decentralized Sybil defense mechanisms, and 5) AI-driven automation with large language models (LLMs) to uncover smart contract vulnerabilities. Collectively, these approaches are designed to detect, mitigate, and prevent security attacks and vulnerabilities in blockchain-based IoT, IIoT, and Metaverse environments, thereby enhancing the overall security and resilience of these systems.

3.1 Research scope

This thesis addresses critical security challenges in blockchain systems, organized across four core dimensions: consensus algorithms, attack types, blockchain platforms, and defense strategies. The research outcome is presented through seven original articles, each targeting a specific aspect of blockchain security.

From the perspective of consensus mechanisms, this study focuses on three widely adopted protocols: Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA). Regarding attack types, the thesis addresses both network-level attacks and code-level vulnerabilities. The security attacks under investigation include three critical categories of Denial-of-Service (DoS) attacks: eclipse, selfish mining, and Sybil attacks. For code-level threats, the study examines three prevalent vulnerabilities in smart contracts: Reentrancy (RE), Integer Overflow/Underflow (IoU), and Timestamp Dependency (TD). Another dimension of this research concerns the underlying blockchain platforms, specifically targeting both Bitcoin and Ethereum networks. Finally, the scope of this thesis spans multiple defense strategies, including detection, mitigation, and prevention.

The research contributions presented in this thesis have resulted in the following publications:

- Chapter 4 addresses eclipse attacks on a PoW-based Bitcoin network.

- Chapter 5 and 6 investigate selfish mining detection and mitigation strategies on PoW-based Bitcoin networks.
- Chapter 7 proposes detection and prevention mechanisms for Sybil attacks within PoA-based Ethereum networks.
- Chapter 8 and 9 present a novel framework for automated smart contract vulnerability detection on PoS-based Ethereum networks.
- Additionally, Appendix A provides a comprehensive survey on the application of game-theoretic models on blockchain security (presented in the Appendix), grounding the thesis in a systematic theoretical context and informing the design of mitigation strategies.

This categorization highlights the comprehensive coverage of the thesis across various blockchain domains, consensus mechanisms, security concerns, and different defense strategies. The overall research scope, illustrated in Figure 3.1, demonstrates the breadth of the study, encompassing consensus protocols (PoA, PoS, PoW), critical security concerns (network attacks and software vulnerabilities), and various defense strategies (intrusion detection, mitigation, and prevention).

Importantly, this work addresses these challenges within a broad array of application domains, including Bitcoin, Ethereum, decentralized applications (DApps), IoT/IIoT, and Metaverse environments, thereby ensuring the relevance and applicability of the proposed solutions to both established and emerging blockchain platforms.

The contributions and research methods are organized under three main defense categories as illustrated in Figure 3.2. The selection of detection, mitigation, or prevention as the primary defense strategy for each attack in this thesis is determined by the inherent characteristics of the threat and the practical feasibility of countermeasures. For eclipse attacks, prevention typically necessitates fundamental modifications to network protocols, rendering it impractical in many deployed systems; therefore, timely detection is prioritized to identify isolation attempts before substantial impact occurs.

Selfish mining is addressed through mitigation rather than prevention, as effective prevention would require altering the consensus mechanism, whereas game-theoretic incentive structures can be applied within existing protocols to diminish the attack's profitability. In contrast, Sybil attacks and smart contract vulnerabilities are approached through prevention, as these threats can be proactively neutralized using decentralized trust mechanisms and automated vulnerability detection prior to exploitation.

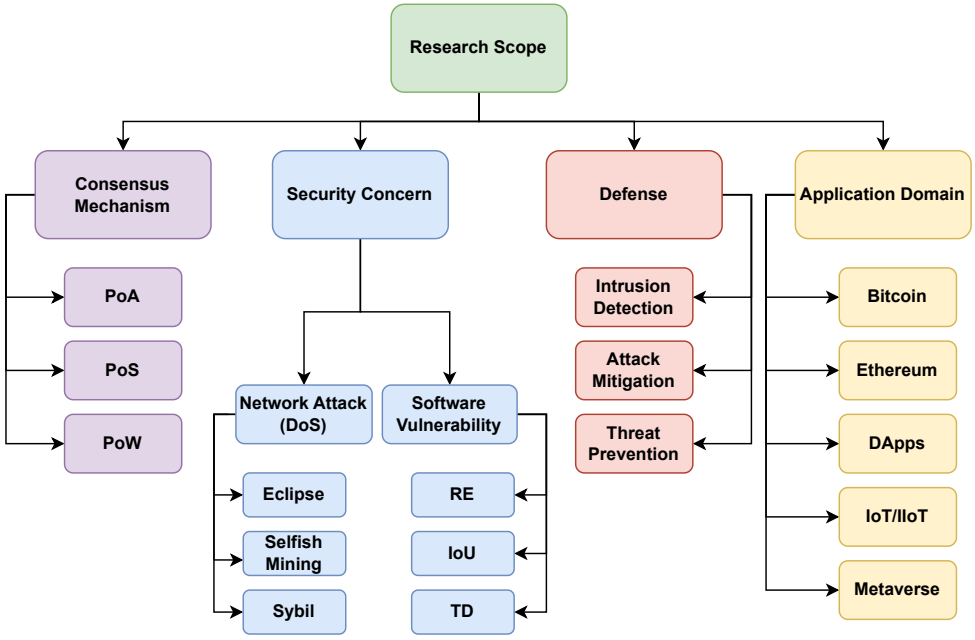


Figure 3.1 Thesis scope.

- **Intrusion Detection:** Strategies for identifying security attacks. This includes the use of graph-based community detection and machine learning to detect attacks, including eclipse attacks in Metaverse systems and selfish mining attacks in IoT-based systems, respectively.
- **Attack Mitigation:** Approaches for reducing the impact of launching security attacks. For instance, the game-theoretic model is utilized to limit the incentives for launching selfish mining attacks on blockchain-based IoT systems.
- **Threat Prevention:** Mechanisms for proactively securing the system against potential attacks. This thesis introduces two complementary approaches to proactive system security. The first approach secures Ethereum-based IIoT networks by integrating a smart contract-based reputation management system with decentralized federated learning (DFL), effectively preventing Sybil attacks. The second approach leverages LLMs to automatically detect critical smart contract vulnerabilities, including reentrancy, integer overflow/underflow, and timestamp dependency, thereby preventing attackers from exploiting these weaknesses and launching security attacks.

These methodological categories not only reflect the strategic flow of this thesis but also serve as a bridge between the research objectives outlined in Chapter 1 and the contributions

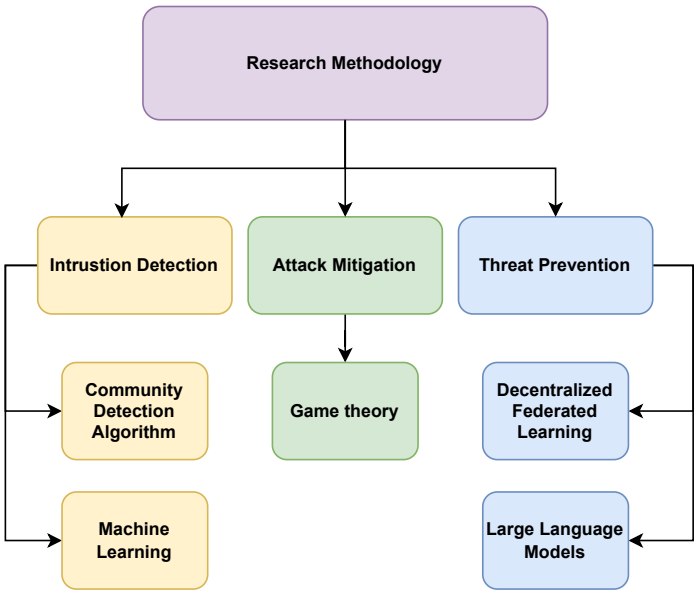


Figure 3.2 The methods and approaches used in the thesis.

presented across the included articles. Each category corresponds to one or more objectives, supported by dedicated studies that show how the selected techniques are applied in practice to detect, mitigate, and prevent attacks in various blockchain environments.

To clearly demonstrate how the research problems, objectives, methodology, and contributions are interconnected, Table 3.1 presents a comprehensive mapping between these components. This mapping ensures the coherence of the thesis structure and the alignment between the problem space and the proposed solutions.

In addition to our direct contributions in threat detection, mitigation, and prevention, this thesis includes a foundational survey that explores the application of game theory to blockchain-based IoT networks. This survey informs the selection of appropriate models, particularly the Retaliation Game used for selfish mining mitigation (Objective #3).

3.2 Overview of approaches and methods

This section summarizes the methodological implementation of the key contributions listed in the previous chapter. Each subsection outlines the techniques adopted to detect, mitigate, or prevent security attacks in various blockchain environments, highlighting the rationale behind the selected methods, their execution, and validation. For detailed experimental results and technical proofs, the reader is referred to the corresponding research articles in this thesis.

Beyond the main contributions related to detection, mitigation, and prevention, this thesis also presents several foundational efforts to enhance the reproducibility, applicability, and theoretical underpinnings of the proposed security framework. These include a comprehensive survey of game-theoretic models applied to blockchain-based IoT systems, the fine-tuning of a large language model for identifying vulnerabilities in smart contracts, the development of a real-time auditing plugin for VSCode to assist developers, and the release of a publicly available smart contract vulnerability dataset. These contributions collectively support the core research objectives while offering practical tools and insights to guide future work in this area.

3.2.1 Intrusion detection

Under the first defense category, we target the detection of eclipse attacks on the Metaverse as well as selfish mining attacks on IoT networks.

Scalable detection of Eclipse attacks on Metaverse

Eclipse attacks pose a critical threat to blockchain-based Metaverse environments by isolating a node from the rest of the network, potentially enabling further exploits such as double-spending. To counter this, we propose a lightweight intrusion detection system (IDS) based on graph analysis and community detection to identify such attacks early and efficiently. Our method consists of two phases. In the first phase, protocol-specific messages (e.g., *INV, ADDR, VerAck*) are captured and transformed into eight message-specific communication graphs representing peer interactions. In the second phase, the Leiden community detection

Table 3.1 Mapping of research objectives and methods.

Research Question	Research Objective	Defense Category	Approaches and Methods	Application Domain
RQ1	Scalable detection of eclipse attacks (Objective #1)	Detection	Chapter 4: CDA (Leiden algorithm)	Metaverse
RQ2	Efficient detection of selfish mining attacks (Objective #2)	Detection	Chapter 5: Machine Learning (RFC and PCA)	Bitcoin
RQ3	Selfish mining mitigation (Objective #3)	Mitigation	Chapter 6: Retaliation Game (war of attrition)	Ethereum/Bitcoin and IoT
RQ4	Decentralized Sybil attack prevention (Objective #4)	Prevention	Chapter 7: DFL, reputation system	Ethereum/IIoT
RQ5	Smart contract security (Objective #5)	Prevention	Chapters 8 and 9 : LLM-based analysis, Fine-Tuning, VSCode Plugin	Ethereum/DApps

algorithm is independently applied to each graph. The overlap of communities surrounding the victim node is analyzed to identify coordinated attacker clusters.

We implemented this approach in a simulated Bitcoin network using an extended version of the Zelig simulator. Our system achieved a detection accuracy of 91.54% and a specificity of 98.98%, while maintaining low overhead, making it suitable for deployment in resource-constrained blockchain nodes. The approach's reliance on protocol-level patterns and its selective detection logic makes it scalable and adaptable to evolving network topologies. Full methodological details and results are presented in Chapter 4.

Rationale behind using the Leiden Algorithm: We selected the Leiden algorithm as our community detection method due to its strong performance in identifying well-connected, non-overlapping clusters in large-scale networks. Unlike older algorithms such as Louvain, Leiden guarantees communities that are both internally connected and globally coherent, reducing the risk of fragmented or misleading groupings. This property is particularly important in our context, where we aim to detect colluding attacker groups surrounding a victim node. As demonstrated in social network analysis, the Leiden algorithm offers fast convergence, scalability, and high-quality community structures, making it a suitable choice for detecting malicious subgroups in decentralized blockchain environments.

Efficient selfish mining attack detection in IoT networks

Selfish mining attacks disrupt the consensus process in blockchain-based IoT systems by allowing malicious miners to gain disproportionate rewards through block withholding. To address this threat, we propose an efficient machine learning-based detection framework that combines Principal Component Analysis (PCA) with Random Forest Classification (RFC). In addition, we present a taxonomy of existing approaches related to selfish mining attack detection to situate our methodology within the broader blockchain security research.

Our methodology begins with dimensionality reduction using PCA to isolate the most informative behavioral indicators of selfish mining. Two datasets are used: one from an existing benchmark and another generated using a simulated Bitcoin network in the Gervais NS-3 simulator. Feature extraction focuses on network-level indicators such as block reception times and propagation delays, ensuring simulator independence.

Following preprocessing and normalization, the RFC model is trained and optimized using GridSearchCV and 5-fold cross-validation. The final classifier achieves a detection accuracy of 99.96%, outperforming the fully connected neural network (FC-NN) used in the ForkDec approach, which reaches 98.98% accuracy but incurs significantly higher computational over-

head. Compared to ForkDec, our RFC-based method reduces training time by approximately 75%, making it a more efficient and practical solution for resource-constrained environments such as IoT-based blockchain systems.

Rationale behind choosing RFC: This lightweight detection framework is particularly well-suited for resource-constrained IoT environments and demonstrates the feasibility of interpretable and efficient learning-based security mechanisms within blockchain networks. While state-of-the-art frameworks often rely on complex and computationally intensive deep learning models, our approach achieves higher accuracy using a more lightweight solution. Full implementation details and performance analysis are provided in Chapter 5.

3.2.2 Attack mitigation

Under the second defense category, we target the mitigation of selfish mining attacks on blockchain-based IoT networks.

Retaliation game for selfish mining attack mitigation

Selfish mining attacks seriously threaten blockchain networks by enabling malicious miners to withhold newly mined blocks and gain disproportionate rewards, while honest miners unknowingly waste their resources. This study proposes a retaliation game framework, modeled as a war of attrition, to discourage such behavior by integrating a reward-punishment mechanism and a reputation system.

The approach first determines the optimal point at which honest miners should stop retaliating against selfish behavior, thereby minimizing their own losses. It then reduces miner indifference by introducing a reputation model that disincentivizes participation in mining pools with known selfish actors. Miners who fail to broadcast their blocks promptly are penalized, discouraging the adoption of selfish strategies.

By combining these mechanisms, the model ensures that withholding blocks become unprofitable over time. An evolutionary game-theoretic analysis is conducted to study the dynamics of strategy selection among miners. Results show that the balance between rewards and punishments is critical in shaping behavior. As more miners choose to reveal their blocks, the network converges to a stable equilibrium where honest strategies dominate. This game-theoretic mitigation framework effectively reduces the incentive for selfish mining and improves blockchain security. The details are available in Chapter 6.

Rationale behind using game theory: Game theory is particularly well-suited for modeling the strategic behavior and interactions of participants in decentralized blockchain net-

works. In this study, we employ the retaliation game, a variant rooted in the war of attrition model, to reflect the realistic decision-making process of miners. This framework captures the ongoing tension between honest and selfish miners, where each participant continuously evaluates the trade-offs between adhering to honest behavior and engaging in malicious strategies to maximize rewards. By modeling these dynamics, the retaliation game facilitates the design of adaptive reward and punishment mechanisms that evolve in response to miner behavior. Unlike static defense strategies, game-theoretic approaches account for the strategic adaptation of players over time, offering a more robust and sustainable solution to mitigating selfish mining attacks.

Game-theoretic models for blockchain security

To support the selection and understanding of game-theoretic strategies for blockchain security, we conducted a comprehensive survey of game theory applications in blockchain-based IoT networks. This survey examines a wide array of game-theoretic models—including Stackelberg, repeated, Bayesian, and differential games—applied to address security threats such as selfish mining, majority attacks, and resource misuse, as well as challenges in consensus, communication, and energy optimization.

Our taxonomy organizes these models according to their application domains (e.g., security, communication, consensus mechanisms) and identifies gaps in the literature, such as limited implementation in real-world systems and underexplored threats like eclipse attacks. The analysis highlights key assumptions and limitations of current approaches while outlining directions for adaptive, incentive-aligned security frameworks in adversarial settings.

This foundational work informs the theoretical basis of our mitigation efforts by offering a structured lens through which game-theoretic mechanisms, like our retaliation game for selfish mining, can be selected and justified. It also serves as a valuable resource for researchers designing secure, resource-aware blockchain-based IoT systems.

3.2.3 Threat prevention

In the third defense category, we address Sybil attacks in blockchain-enabled IIoT environments as well as smart contract vulnerabilities to prevent attackers from launching attacks and exploiting these vulnerabilities in DApps.

Sybil prevention using DFL and reputation management

Sybil attacks threaten the security and reliability of blockchain-enabled IIoT systems by allowing adversaries to inject numerous fake identities, undermining trust, disrupting network operations, and enabling subsequent exploits such as data spoofing or denial-of-service. To address this challenge, a scalable prevention architecture is introduced that combines DFL with smart contract-driven reputation management on an Ethereum PoA blockchain. This approach is designed to ensure data privacy, system robustness, and real-time Sybil prevention in industrial environments.

The methodology consists of two core phases. In the detection phase, an MQTT broker—acting as both an IIoT gateway and Ethereum node—monitors multi-protocol network traffic (including MQTT, devp2p, discovery, and ENR) and extracts 16 behavioral features from observed communication patterns. Each broker locally trains a convolutional neural network (CNN) for Sybil detection using these features. Model weights are exchanged directly among peer brokers without relying on a central coordinator, following a decentralized aggregation strategy. This DFL framework ensures that sensitive local data remains private while eliminating single points of failure inherent in centralized learning.

In the prevention phase, brokers submit digitally signed severity reports—summarizing detected anomalies—to a reputation management smart contract deployed on the PoA Ethereum network. This contract dynamically adjusts reputation scores, automatically blacklisting nodes whose scores fall below a configurable threshold. The threshold value is determined through empirical tuning on validation data, selecting the point that maximizes detection accuracy while minimizing false positives. The entire process is auditable and tamper-resistant, supporting adaptive trust management and real-time exclusion of suspicious nodes from the IIoT system.

We evaluated this defense system in a simulated PoA-Ethereum IIoT network with 300 nodes, generating a comprehensive labeled dataset of Sybil and honest behaviors. Our approach achieved average detection accuracy and recall rates above 91% across clients, while preserving privacy and maintaining resilience to both Sybil infiltration and model poisoning. Compared to traditional federated or centralized deep learning, the DFL-based method demonstrated superior privacy, trust scalability, and attack resistance.

Rationale behind leveraging DFL: DFL was selected for its ability to provide robust, privacy-preserving intelligence in distributed IIoT settings. Unlike standard federated learning, which still depends on a central aggregation server (thus introducing a single point of failure), DFL fully decentralizes model training and aggregation. This architecture ensures

that raw industrial data never leaves the local node, addressing strict privacy requirements, and enables the system to remain operational even if individual brokers are compromised or disconnected. Moreover, DFL's peer-to-peer (P2P) model exchange improves robustness against adversarial manipulation and supports heterogeneous IIoT deployments. The integration of DFL with blockchain-based reputation management offers a comprehensive, real-time defense against Sybil attacks, tailored for the stringent needs of industrial blockchain environments.

Design of Sybil simulator

To support experimentation and model evaluation, we designed and implemented a dedicated Sybil attack simulation framework specifically for PoA-based Ethereum IIoT environments. This simulator accurately mimics the complex, real-world interplay of multiple network protocols—including MQTT, devp2p, discovery, libp2p, and ENR—found in industrial settings. By capturing detailed communication logs across these layers, the simulator enables automated extraction of key behavioral features, facilitating the creation of a comprehensive, labeled dataset that differentiates between honest and Sybil node activities.

Full methodological details and evaluation results are provided in Chapter 7.

Automated identification of smart contract vulnerabilities

This work introduces a novel vulnerability detection approach leveraging large language models (LLMs) to accurately identify, explain, and fix smart contract vulnerabilities—specifically integer overflow/underflow (IoU), reentrancy (RE), and timestamp dependency (TD). Due to the bias introduced by comments and function names in the original SolidiFi dataset, we first performed a thorough preprocessing phase, removing such cues to ensure a realistic and unbiased evaluation.

The methodology centers on a chain-of-thought (CoT) prompting strategy guided by expert patterns. The prompt design includes system instructions, few-shot examples, and structured outputs, enabling the model to output vulnerable lines of code, reasoning, attack potential, and corrected code. Two GPT-4o models were tested (GPT-4o and GPT-4o mini), both of which outperformed existing static and dynamic tools in recall and F1 score while maintaining lower false positive and false negative rates.

The results show high effectiveness, with GPT-4o achieving recall scores of 93.5% (IoU), 95.4% (RE), and 93.8% (TD). This method significantly outperforms traditional tools such as Slither, SmartCheck, Mythril, and even recent LLM-based methods, delivering more reliable

vulnerability auditing with minimal manual intervention. The final dataset and codebase are publicly available to support future research. The details of this approach are stated in Chapter 8

Rationale behind using LLMs: The use of LLMs is motivated by their strong capabilities in understanding and reasoning over code syntax and semantics. Unlike traditional tools relying on predefined rules and symbolic execution, LLMs can generalize across diverse code patterns and vulnerabilities. In particular, chain-of-thought prompting guides the model through a step-by-step reasoning process, mimicking expert-level vulnerability analysis. This structured reasoning enables the model not only to detect subtle and context-dependent vulnerabilities but also to explain the underlying logic, assess the exploitability, and generate corrected code. By incorporating CoT, the detection process becomes more transparent and interpretable, reducing the likelihood of false positives and enhancing trust in automated analysis.

Real-Time VSCode plugin tool for vulnerability detection

We also introduce a real-time, practical, and reliable VSCode plugin designed for smart contract vulnerability detection. The tool is open-source, allowing developers to contribute to its development and extend its functionality. Built on LLMs, particularly GPT-4o, the plugin also supports model customization, enabling users to switch to alternative GPT-based models if desired. This tool offers valuable assistance to smart contract developers by integrating automated security analysis directly into their development environment. It is publicly available and freely accessible, aiming to support both the research community and industry practitioners.

LLM fine-tuning for smart contract vulnerability detection

As discussed in Chapter 8, we proposed a framework leveraging GPT models to detect smart contract vulnerabilities. Building upon this foundation, the contribution presented in Chapter 9 focuses on fine-tuning a large language model to create a customizable and locally deployable solution for developers. Specifically, we fine-tune *LLaMA-3.1-8B* using *LoRA* for memory-efficient training, enabling the development of a standalone, self-hosted security auditing model that operates independently of proprietary APIs. In contrast to our VSCode plugin, which remains API-dependent, this model offers a cost-effective, scalable, and privacy-preserving alternative tailored for practical deployment in development environments.

Dataset preparation and public release

One of the primary challenges in security research is the lack of comprehensive, reliable, and publicly available datasets. To address this gap in the smart contract vulnerability detection domain, we have developed, curated, and preprocessed a robust dataset to support research and evaluation efforts. This dataset is designed to be comprehensive and well-structured, making it suitable for a wide range of experimental and benchmarking purposes.

3.3 Conclusion

This chapter outlined the research methodologies adopted in this thesis, which address the detection, mitigation, and prevention of security attacks—including eclipse, selfish mining, and Sybil attacks—as well as the detection of critical vulnerabilities in Ethereum-based smart contracts. The following chapters present the peer-reviewed articles that contain each of these contributions.

CHAPTER 4 ARTICLE 1: COMMUNITY DETECTION ALGORITHM FOR MITIGATING ECLIPSE ATTACKS ON BLOCKCHAIN-ENABLED METAVERSE

Fatemeh Erfan, Martine Bellaiche, Talal Halabi

Published on IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom), Kyoto, Japan, pp. 403–407, June 26, 2023

Abstract

The Metaverse has recently attracted a lot of attention from academia and the industry. Blockchain technology is a critical coordinator for the Metaverse, providing a secure and decentralized infrastructure to create and manage digital assets. Nonetheless, several security threats arise from using blockchain in the Metaverse such as Distributed Denial of Service (DDoS) in peer-to-peer networks. The eclipse attack is one of the most common and easy-to-launch DDoS attacks on blockchain networks and can further lead to 51%, selfish mining, double-spending, and race attacks. However, eclipse detection remains a challenge for light nodes that cannot deploy complex intrusion detection systems. In this paper, we design an intrusion detection system based on a community detection algorithm (CDA) to detect every attempt to launch an eclipse attack through blockchain-enabled metaverse systems. First, the blockchain traffic data is captured from peers participating in the blockchain network. Then, one of the most efficient CDAs, namely the Leiden algorithm, is applied to detect eclipse attempts. The experimental results show that the proposed approach can identify the potential malicious nodes with high accuracy in order to monitor their behaviors and connections through the network and mitigate the attack.

Keywords: Eclipse attack, blockchain networks, Metaverse, community detection algorithms, intrusion detection system.

4.1 Introduction

Metaverse, the next iteration of the Internet, refers to a virtual environment where individuals can attend diverse activities (e.g., socializing, gaming, virtual events, education, and business) through their digital representation, known as avatars. Metaverse integrates various technologies such as Virtual Reality (VR), Augmented Reality (AR), Artificial Intelligence (AI), and blockchain [48]. The latter can play a vital role in the functioning of the Metaverse by offering a secure and decentralized framework for creating and managing digital assets

and recording transactions across the network without relying on third parties [20]. The Metaverse gathers tremendous personal information to enhance user experience. However, if the data is exposed to unauthorized parties, it could result in real-world targeting. Hence, blockchain technology gives users full control over their data, ensuring their privacy is protected.

The blockchain as a distributed ledger based on a peer-to-peer (p2p) network has drawn tremendous attention due to its salient features such as transparency, quality assurance, and decentralization, which enable a high level of security making it difficult to manipulate data or subvert the network. Nonetheless, blockchain security has recently been questioned [65]. One of the most challenging cyber attacks launched against blockchain is the eclipse attack proposed by Heilman et al. [52] which allows an attacker to control all the connections from and to the victim node by monopolizing several IP addresses and filtering the node's view of the blockchain. Hence, the victim cannot see or transact on the honest network as it becomes isolated from the rest of the network. Thus, all its transactions will be registered only on the ledger under the attackers' control and not the original blockchain [132]. This attack can impact the stability of the whole blockchain network [53] and provide the preconditions for launching other attacks, including selfish mining, majority, Denial of Service (DoS), double-spending, and engineering block races attacks [53].

Eclipse can be considered a form of Distributed DoS (DDoS) attack, where several nodes overwhelm a victim's server with a large amount of traffic, causing it to become unavailable. In the context of the Metaverse, the attack could have notable consequences as it may cause widespread disruptions to the virtual world and prevent users from participating in virtual activities and accessing their virtual assets. Eclipse attack traffic behavior features are almost similar to regular traffic. Thus, distinguishing between malicious and honest traffic based on the extracted features remains challenging. There are a few existing works using machine learning and deep learning to detect the eclipse attack through blockchain networks [53, 130–133]. However, they suffer from generalization since they do not provide real-time detection and cannot detect the attackers.

This paper proposes a novel approach to detect the eclipse attack against a blockchain-based network in Metaverse by relying on the concept of community detection. Community detection algorithms (CDA) have attracted a lot of attention in computer science and mathematics thanks to their potential applications to communication networks, social networks [77, 78], and collaboration networks [79]. For instance, community detection can simplify recommender systems on social media, including Facebook and Twitter, by finding the groups of users closer to each other [77]. As CDA plays a significant role in discovering which connected nodes have more in common, it can be leveraged in blockchain systems to detect malicious

nodes or groups. To the best of our knowledge, this is the first work that leverages CDA to mitigate security attacks on blockchain networks, which will increase the protection of blockchain-enabled Metaverse systems against DDoS attacks.
Our contributions are described as follows. We explore the bridge between *blockchain security* and *community detection* by applying one of the most efficient CDAs to blockchain networks to generate communities based on indicator packets related to the eclipse attack. Our CDA-enabled approach can efficiently detect and mitigate eclipse attacks launched on lightweight nodes or gateways in Metaverse networks by identifying potential malicious nodes monopolizing the victim's connection table. We implement and evaluate the security and performance of our detection scheme using a bitcoin network.

The remainder of the paper is organized as follows. Section 4.2 discusses background concepts and related work. Section 4.3 describes the proposed approach for detecting and mitigating eclipse attacks on blockchain networks. Section 4.4 presents the implementation details and analyzes the experimental results. Finally, Section 4.5 concludes the paper.

4.2 Background

This section reviews background information related to Metaverse security and community detection algorithms.

4.2.1 Metaverse and Network Security

The infrastructure of Metaverse is built upon technologies such as AI, distributed computing, and blockchain. This paper focuses on blockchain and its network security. Based on [171], the Metaverse system has a service provider, i.e., Metaverse Service Provider (MSP), that proposes virtual services and applications such as conferences, games, and events to the Metaverse Users (MU). Each MU can create an account with digital assets, including coins, properties, and tokens, and trade with the MSP to access specific services. Blockchain can be a potential framework for handling the relations between MSP and MU as well as among MUs so that all their interactions are stored in the blockchain as transactions.
Data privacy and network security pose significant risks in the Metaverse. A large amount of data gathered by users is bound to the Metaverse, which can lead to challenges in tracing malicious behavior and increase the likelihood of security and trust issues [172]. There are several network security threats in the Metaverse, including DDoS and Sybil [47]. MU devices can be targeted by attackers to form botnets and launch DDoS attacks. Eclipse is a type of DDoS where an attacker controls some nodes to block the victim's view from the

rest of the Metaverse network by manipulating their IP address tables.
Centralized methods cannot handle the large-scale data recording, sharing, and storage in the Metaverse due to high computation and communication overheads. Blockchain can tackle these challenges and guarantee the reliability and consistency of Metaverse transactions thanks to its cryptographic foundations [172]. We focus on the network layer in the blockchain, which may be susceptible to eclipse by exploiting the vulnerability of routing in P2P protocols. In the Metaverse, the situational awareness tool is responsible for monitoring and detecting threats and anomalies [173]. Hence, the eclipse detection approach proposed in this paper can be integrated into this tool to protect blockchain-enabled Metaverse networks.

4.2.2 Community Detection Approaches

Community detection is used to discover groups of nodes and extract them as communities. When analyzing networks, extracting communities by investigating the connections among the nodes may be valuable [81, 82]. A community can be defined as a subset of the nodes with more robust connections between peers inside the community and weaker connections between peers outside the community [83].

Definition 1. Modularity: Measures the quality of dividing nodes into different communities [174]. If the number of edges in a cluster is not better than the state of the random graph, the modularity is equal to zero. Its maximum value is obtained when all the vertices of each cluster are connected, and no edge connects the clusters. The modularity is given by [175]:

$$H = \frac{1}{2m}\sum_{c}(e_c - \gamma\frac{K_c^2}{2m}) \tag{4.1}$$

where $\gamma > 0$ is a parameter defining the resolution of the community detection method. The higher is the resolution, the more communities are detected. In community c, e_c is the real number of edges. $\frac{K_c^2}{2m}$ is the anticipated number of edges in community c. The total degrees of all nodes in community c is represented as K_c. The sum of all edges in the network is stated as m. This formula maximizes the difference between the real and the anticipated number of edges.
Definition 2. Network graph: Let $G = (V, E, W)$ be a weighted, directed graph where V are the vertices (the number of nodes participating in the network is n), E are the edges of the graph which represent the connections between nodes, and W are the weights of each edge which is the number of messages transmitted between two vertices V_1 and V_2.
Definition 3. Community detection: By applying CDA such as the Leiden algorithm L_i to the graph G, the communities can be detected as $C = \{C_1, C_2, \ldots, C_k\}$. In the network G,

the approach of assigning each node v_i into one community $C_i \in \{C_1, C_2, \ldots, C_k\}$ is called community detection. Community detection approaches are divided into agglomerative and Divisive based on whether the algorithm is adding or removing edges to or from the network [174].

Leiden algorithm: Before the Leiden algorithm [175], the Louvain algorithm [176] was the most popular for finding communities in networks. Traag et al. [175] presented the Leiden algorithm by highlighting the main problems of the Louvain algorithm such as finding communities where the nodes are not or poorly connected. The algorithm benefits from the idea of moving nodes to random neighbors [175]. Finding hidden communities can have drastic advantages. It can provide simple representations of networks and the complex relations among entities. We argue that it can also be used to discover malicious groups inside blockchain networks.

4.2.3 Eclipse Attack and Detection

With the openness of blockchain platforms, it is easy for a malicious node to join the network and attempt to launch an eclipse attack by monopolizing all incoming and outgoing connections of the victim node [52]. The latter receives rapidly and repeatedly undemanding requests from the malicious node and false information about the ledger from compromised nodes, which try to fill the victim's table by replacing the IP addresses of honest nodes with their own. Detection approaches [53, 130–133] either rely on routing topology perceptions (changes in the routing state) or link traffic state analysis by monitoring the real-time traffic status to find the features of eclipse traffic [133].

Xu et al. [131] proposed detecting an eclipse attack launched on Ethereum using prediction models by analyzing network packets based on random forest classification. Dai et al. [133] also used a custom combination of features and deep learning techniques. In their solution, Alangot et al. [132] considered a trade-off between the network load and detection time. Although these approaches attempt to mitigate eclipse attacks, they may not be suitable for large-scale, low-power nodes since they yield a substantial computational overhead and do not offer real-time detection. To the best of our knowledge, our approach is the first that utilizes community detection techniques to predict and detect efficiently the eclipse attacks in blockchain networks. This approach can be more appropriate for resource-constrained nodes in future Metaverse networks.

4.3 Proposed Approach

We design an Intrusion Detection System (IDS) for blockchain-enabled Metaverse involving two phases: 1) creating directed-weighted graphs of the network using captured data, and 2) applying CDA to the network graphs.

4.3.1 System Model

The main system entities participating in this detection approach are described as follows:
Honest node: blockchain nodes consisting of full, light, and null nodes. Full nodes are responsible for saving the full blockchain ledger, while Light nodes rely on full nodes as they can only save block headers. Null nodes are only responsible for transmitting blocks through the network. As nodes can be resource-limited, we consider them light or null nodes.
Malicious node: intends to create a malicious group, including several dishonest nodes that try to manipulate the victim's table and launch the eclipse attack.
IDS unit: monitors and analyzes network traffic and nodes' activities. If there is abnormal traffic, it activates the attack detection module and identifies the malicious nodes.
The following requirements are critical for the effectiveness and deployment of our IDS in large-scale networks of resource-constrained nodes: 1) Accuracy: we aim to design an IDS to detect the eclipse attack with high probability; 2) Low complexity and high efficiency: the computation, communication, and storage overheads of the IDS should be minimal; and 3) High adaptability: the IDS should be compatible with any blockchain network and should not need substantial changes for setup or execution.

4.3.2 Threat Model

We assume the malicious node cannot break the secure channel between a blockchain node and the IDS. However, it has sufficient computation and communication resources to launch an eclipse DDoS attack. The attacker controls a group of nodes and turns them into malicious nodes, e.g., using malware. The threat model assumes that only one victim is targeted in the network. Before the eclipse, the victim can get the ledger data from other nodes in the network. As shown in Figure 4.1, during the eclipse attack, the victim only connects to the group of malicious nodes and cannot communicate with other nodes since its view of the blockchain network is eclipsed.
In our network, each peer can send a request to get the list of all peers involved in the network. Each node has at most 125 active connections to other peers. If a node finds a new block, it can broadcast a message containing digests of the found block. The bitcoin network

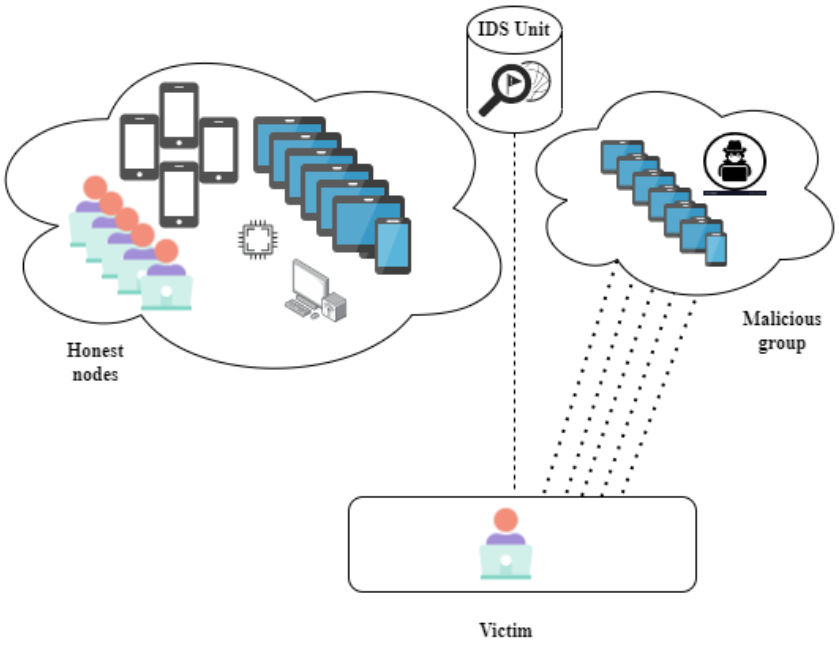


Figure 4.1 System model under the eclipse attack.

is a public network, and every node can receive the transmitted message. If the node figures out that the blocks are already received, the node ignores this message. If the digest of the block is new, the node requests a message by sending a *GETDATA* message.
Every node has two tables named tried and new. These tables include the IP addresses of nodes participating in the network. The tried table of node A consists of the IP addresses of all nodes connected to A at least once. The IP addresses in the new table refer to the nodes that have never established any connection to node A. Each peer updates its known IP addresses in its tried table, uninterruptedly. If the node can establish a new connection to the nodes in the new table, its IP addresses will be moved from the new to the tried table.

For instance, if the malicious node connects to the victim, the latter receives unwanted ADDR messages and stores them in its new table, and the old IP addresses will be moved to tried tables. Thus, the legitimate IP addresses stored in the victim's tried table will be overwritten. In our scenario, the attacker uses 31 nodes and puts their IP addresses into the victim's tried table. When the attackers populate the victim's tables, the latter cannot communicate with other peers to get the correct information about the ledger.

4.3.3 Eclipse Detection Model

Phase I: Data Collection The bitcoin network's transmitted flows are monitored and captured after generating blocks. There are eight types of messages captured that are essential

for eclipse detection on the bitcoin network [52, 131, 133]:

INV: Sent by a node participating in the bitcoin network to announce its block that has been mined.

Version: Sent by a node to open an outgoing connection.

VerAck: when a node receives a *Version* message, it replies with a *VerAck* message that includes its own bitcoin protocol.

GetADDR: when a node wants to get a list of addresses that others know, it should send this message.

ADDR: sent as a response to the *GetADDR message*. *GetData:* is the response to the *INV* message.

Ping: when the node wants to know if a peer in its list is still online, it sends this message.

Pong: this message is sent to reply to the *ping message*.

The IDS unit can capture, analyze, and store these messages. The output of the first phase is eight graphs denoted by $G(i)$, $i = 1, \ldots, 8$, based on the types of captured messages. The details of the first and second phases of the proposed approach are explained in Algorithm 1.

Phase II: Attack Detection First, the network is represented as eight directed weighted graphs based on eight types of messages captured in the previous phase. It is noticeable that the degree of graphs represents node connections. Here, we apply the Leiden algorithm $L(i),\ i = 0, \ldots, 8$ to the eight graphs in order to find the communities through the network. We are looking for all nodes in the same community as the victim, and we assume them as

Algorithm 1 Data collection & Eclipse attack detection

Phase I: Data collection request triggered by the victim node
Declaration:
  $L(i)$: The list of nodes in the same communities as the victim
  INC: The list of $L(i)$
  $Packets(i) \in$ {VerAck, Ping, Pong, GetData, GetAddR, INV, Version, Address}
Input: victim's IP address
for i **do**
  Sniff $Packets(i)$
  Packets.append($Packets(i)$)
end for
Phase II: Attack Detection by the IDS
Input: Packets
for all i **do**
  Create a graph $G(i)$
  $L(i) \leftarrow$ Apply Leiden Algorithm over $G(i)$
end for
for i **do**
  INC.append($L(i)$)
end for
INC-Final $\leftarrow$ intersects(INC)
if INC-Final is not zero **then**
  return intersects(INC)
else
  return "Not Detected"
end if

the victim's list. Then, if the intersection *Inc* between these eight graphs $G(i),\ i = 0, \ldots, 8$ is a null set, then there is no malicious traffic detected. In other words, no attack is launched or no attacker or a group of attackers are trying to launch an eclipse attack.
Furthermore, forming communities in blockchain networks can help alleviate the communication and computation overhead caused by the real-time data collection and traffic monitoring performed by the IDS in large-scale environments. This is because the IDS can now identify potential malicious communities where it can closely monitor the behavior of suspicious nodes in smaller peer groups.

Runtime Scenario In the bitcoin network, if a node is eclipsed by an attacker or a group of attackers, it cannot figure out if it is under attack or not without asking another unit, such as the IDS, since the attacker controls the node's view of the network. We assume that there is a blockchain-enabled Metaverse system, including several nodes connected through the bitcoin blockchain. Assuming that one node suspects to be eclipsed, it sends a request through a secure channel to the IDS unit, which then runs Algorithm 1. After data collection, it creates a network graph and applies the Leiden algorithm (phase II). Then, the IDS starts to look for all nodes in the same community as the victim. After that, the IDS unit intersects with the results. If the output is zero, no attack was launched or is being established. Note that this IDS unit not only can detect the potential eclipse attack but also can find the potential malicious nodes which are trying to send multiple messages to monopolize the victim's tables.

4.4 Experimental Results

In this section, we describe the implementation details and analyze the results. We simulate a bitcoin network by modifying the Zelig simulator [177] on a virtual machine running Linux OS 18.04.6 LTS with 11th Gen Intel(R) Core(TM) i7, 16 GB RAM, 2.8 GHz. In phase I, we analyze the bitcoin network, including 120 nodes, and launch an eclipse attack on the network. First, we create the peers, including honest, null, and malicious nodes. There is one victim node, 32 malicious nodes, and 87 honest nodes. We customized the Zelig simulator by adding some features for capturing specific packets, which are important in the eclipse attack detection.
We added the packet observation part to Zelig that can monitor and count the eight types of packets, including *VerAck, GetADDR, Ping, Pong, Address, INV, Version,* and *GetData*. Once phase I is completed, a graph of the nodes participating in the bitcoin network and all their connections is generated. For phase II, we use the Leidenalg, igraph, and networkx libraries. We execute the attack detection phase on the same virtual machine. There are layouts to illustrate network graphs and their connections, including kamada_kawai, Spiral,

and circular, to better understand the network and monitor connections between the nodes and communities more precisely. The graph of the bitcoin network after applying the CDA is illustrated in Figure 4.2. The victim's ID and port are '64364:13046'; this node is in the orange community. The orange nodes are in the same community as the victim (on the victim's list). We apply the CDA on the network until there is no change in the nodes' communities.

The algorithm can discover 22 among 32 attackers participating in the bitcoin network. As shown in Table 4.1, we evaluate the performance of our detection approach using five metrics, including Precision, Recall, Accuracy, Specificity, and F1 score. Since the light and null nodes are considered as peers with resource constraints, the computation overhead is crucial in IDS design. Our framework divided the complexities between both the victim node and the IDS sides. As reported in Table 4.2, the computation complexity of our proposed approach on the client side is independent of the size of the network ($O(1)$), as the nodes that want to check their status do not execute any computational tasks. The IDS computation overhead is $O(n)$, where n is the number of nodes. The IDS should execute the algorithm in $O(n)$ in both phases I and II.

The communication cost between the victim node and the IDS unit is constant $O(1)$ since the node should send a single request to the IDS to ask for the status check on whether it is eclipsed or not. After the IDS runs the algorithm, a single message returns the victim's status to the node. Also, the storage overhead on the client side is negligible, while the storage complexity on the IDS side is $O(n)$. Finally, the execution time of our approach is 22.26 seconds.

4.5 Conclusion

This paper describes a novel approach to detect an eclipse attack launched on blockchain-enabled Metaverse systems by capturing key messages and applying a community detection algorithm named the Leiden algorithm. In this approach, the IDS can accurately detect the attack and the potential attackers, then shift its focus to monitoring their behavior instead

Table 4.1 Evaluation metrics

Metrics	***Precision***	***Recall***	***Accuracy***	***Specificity***	***F1 score***
Results (%)	95.65	68.75	91.54	98.98	80.1

Table 4.2 Efficiency of the proposed approach

Overhead	***Computation***	***Communication***	***Storage***	***Execution time***
Results	*O(n)*	*O(1)*	*O(n)*	22.26 *sec*

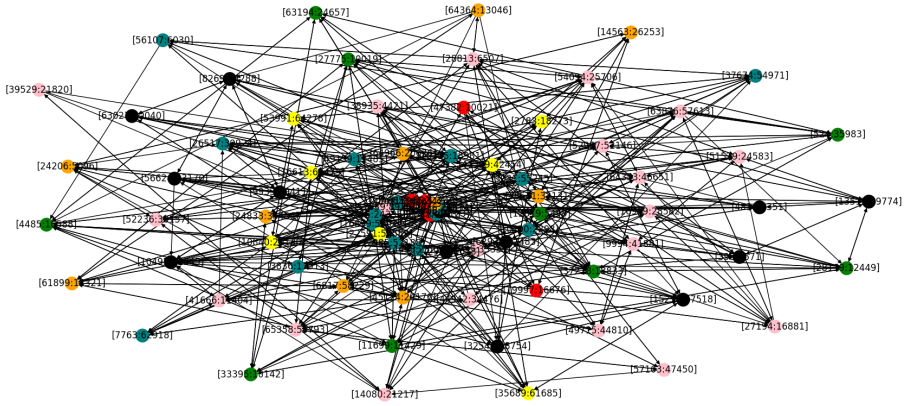


Figure 4.2 The graph of bitcoin network after applying CDA.

of analyzing the whole network. Our detection strategy yields an accuracy and specificity of 91.54% and 98.98% while preserving the limited resources on lightweight nodes. Thus, detecting eclipse attacks without deploying complex and computationally-heavy anomaly detection systems is one of the highlights of the proposed framework. In the future, we plan to use CDA to mitigate other security attacks on the blockchain (e.g., Ethereum), such as selfish mining and Sybil, especially in Metaverse network architectures.

CHAPTER 5 ARTICLE 2: EFFICIENT DETECTION OF SELFISH MINING ATTACKS ON LARGE-SCALE BLOCKCHAIN NETWORKS

Fatemeh Erfan, Martine Bellaiche, Talal Halabi

Published on IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Cambridge, UK, pp. 196–205, July 1, 2024

Abstract

Selfish mining attacks pose a significant and ongoing security threat to blockchain networks, including major platforms like Bitcoin and Ethereum. Understanding and effectively countering these attacks is crucial for maintaining the stability and integrity of these networks. This attack strategy involves a miner or a group of miners seeking to gain an advantage by delaying the immediate broadcasting of their mined blocks to the network. The selfish miners potentially mine more blocks in secret, increasing their rewards at the expense of other miners and the stability of the blockchain. Research has mainly focused on detecting this attack by analyzing data related to blocks and forks. This paper presents a new approach to efficiently detect selfish mining attacks in large-scale networks by analyzing various network indicators. To achieve this, we simulate a Bitcoin network and generate a dataset consisting of several features of individual miners and the overall network. We select the most reliable indicators by analyzing network features and reducing feature dimensions. Then, we employ random forest classification (RFC) to classify benign and selfish network behaviors. This approach not only achieves enhanced accuracy (99.96%), surpassing state-of-the-art methods utilizing deep learning, but also significantly reduces computational, storage, and temporal complexities. In doing so, it fortifies blockchain security more efficiently and accurately.

Keywords: Blockchain, security, selfish mining, machine learning, attack detection, efficiency.

5.1 Introduction

Blockchain is a decentralized system that permits the validation of transactions by a group of untrusted participants [20]. Transactions are collected by the participants, known as miners, who then incorporate them into the public ledger through a consensus protocol. The research on blockchain-enabled systems and their security against cyber threats is experiencing a notable acceleration. Bitcoin [20] and Ethereum [178] stand as prominent examples of blockchain-based applications. Bitcoin is known as the most popular cryptocurrency and utilizes Proof-of-Work, where miners competitively engage in solving a cryptographic puzzle

that is tough to resolve but simple to verify. The miner who successfully cracks the puzzle first can add transactions to the ledger and achieve Bitcoin rewards [54]. Miners collaborate to mine and share new blocks, sometimes resulting in multiple blocks being ahead of the previous one, causing blockchain forks [38]. Hence, the protocol recommends selecting and mining on the longest chain, generally with the most accumulated work [144].

Bitcoin's reward mechanism was traditionally considered incentive-compatible, but the rise of selfish mining challenges this belief [54]. For instance, attackers can strategically release previously withheld blocks to invalidate those mined by honest miners, seizing additional rewards from the honest participants. The consequences of selfish mining extend beyond this immediate impact. An unequal allocation of rewards may motivate rational participants to exhibit selfish behavior. Moreover, a group of selfish miners might collude and join forces to undermine honest participants' revenue, which could tarnish Bitcoin's reputation. In such a situation, it's possible that numerous honest miners might leave the network, significantly reducing its security and opening the door to various other types of attacks, such as double-spending. [179].

In the field of cybersecurity, detection, mitigation, and prevention are three types of approaches that are normally adopted to maintain the security and privacy of a system. In the detection approach, security experts seek to identify attacks and, in some cases, the attackers attempting to manipulate the system's security and compromise its integrity. In the mitigation strategy, while it's not possible to entirely stop attackers from carrying out harmful actions, steps are taken to lower the chances of attacks happening and to lessen their impact if they do occur. On the other hand, in the prevention aspect, specialists focus on actively stopping attackers beforehand, with the goal of decreasing the chances of an attack happening at all. In our paper, we classify our proposed method as part of the detection category since its purpose is to spot selfish mining attacks within a Bitcoin network.

The empirical evidence shows that instances of selfish mining attacks on the Bitcoin Cash network occurred intermittently, with a notable surge in abnormal mining behavior observed in November 2018 and sporadic instances persisting in subsequent years [139]. In 2019, Ethereum Classic, a hard fork derived from the original Ethereum blockchain, also experienced incidents of selfish mining [34]. A recent case involved Monacoin, a Japanese blockchain, resulting in an estimated $90,000 in damages to its network [139]. Therefore, it is imperative to emphasize promptly detecting this attack and implementing countermeasures as soon as it is detected. This paper proposes an efficient approach to detecting selfish mining attacks in large-scale blockchain-based systems like Bitcoin. Our main contributions are listed as follows:

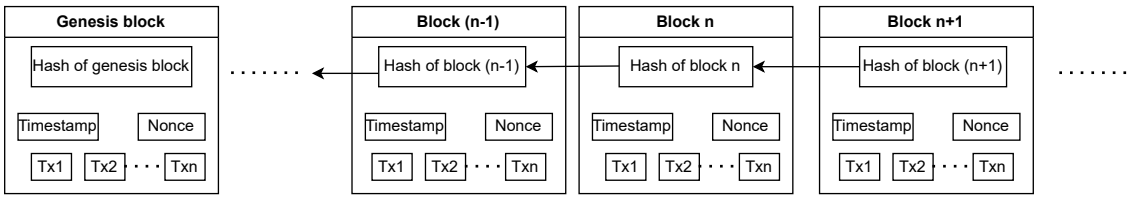


Figure 5.1 The chain of blocks starting from the Genesis block.

- First, we create a new dataset through the simulation of a Bitcoin network, encompassing a comprehensive array of network-related features at both individual node and network-wide levels. This dataset is a valuable tool that can help researchers thoroughly study and improve the detection of selfish mining attacks.
- Second, we propose an enhanced detection methodology for identifying selfish mining attacks in blockchain networks. By leveraging the Random Forest Classifier (RFC), we demonstrate notable improvements in the detection accuracy when compared to an existing deep learning approach known as ForkDec [136]. We employ Principal Component Analysis (PCA) to reduce feature dimensions within the dataset effectively. This dimensionality reduction process contributes significantly to the optimization of the detection model's efficiency and performance.

This paper presents an efficient approach to detecting selfish mining attacks in large-scale, blockchain-based systems like Bitcoin using machine learning (ML). In the dynamic realm of blockchain networks, conventional rule-based methods face challenges adapting to emerging threats like selfish mining attacks. Hence, integrating artificial intelligence (AI) as an analytical tool becomes essential. By leveraging ML, we ensure consistent and reliable decision making when analyzing data patterns. Our goal is to enhance the resilience of blockchain networks against malicious activities. Our main contributions are outlined as follows:

- We investigate the classification of previous research efforts within the domain of selfish mining attacks in blockchain-based systems and provide a taxonomy of existing defense approaches against selfish mining attacks.
- We analyze both node-based and network-based features of selfish mining attacks in the Bitcoin network. Subsequently, we generate a new dataset by simulating a Bitcoin network, encompassing a comprehensive array of network-related features at both individual nodes and network-wide levels. This dataset serves as a valuable tool to aid researchers in thoroughly studying and improving the detection of selfish mining attacks.

- We propose an enhanced detection methodology for identifying selfish mining attacks in blockchain networks. By utilizing the Random Forest Classifier (RFC), we showcase notable improvements in detection accuracy compared to an existing deep learning approach known as ForkDec [136]. We employ Principal Component Analysis (PCA) to effectively reduce feature dimensions within the dataset. This dimensionality reduction process significantly contributes to optimizing the efficiency and performance of the detection model.

Our approach not only provides enhanced accuracy over state-of-the-art methods but also showcases reduced computational, storage, and temporal complexities, thereby fortifying blockchain security in a more efficient and accurate manner.

The remainder of the paper is organized as follows. Section 5.2 presents the background related to blockchain and selfish mining attacks. The related work associated with selfish mining attacks is discussed in Section 5.3. Our methodology is described in Section 5.4. Section 5.5 reports and analyzes our results. Section 5.6 concludes the paper.

5.2 Background Information

5.2.1 Blockchain Concepts

In this section, the fundamental components of a blockchain system are described.

Block: Each transaction sent through the blockchain is stored in a block. Blockchain is a chain of blocks created sequentially by miners. Every block consists of two elements: the header and the body. Transaction data is stored in the body portion of a Merkle tree data structure. Transactions are paired and hashed iteratively until a single hash, referred to as the Merkle Root, is generated [22]. The block header consists of essential data such as the present block's identity, the previous block, a timestamp, a nonce, and the Merkle root. As shown in Figure 5.1, these blocks are linked together, similar to a linked list starting with the first block called the “genesis" block.

Miner: An integral participant in the blockchain network responsible for the process of mining new blocks. When creating a new block, miners follow a series of steps to integrate it into the blockchain. Initially, they work to solve a cryptographic puzzle using a consensus algorithm. Subsequently, the blockchain network members validate the newly created block. Notably, it is unnecessary for miners to maintain a local copy of all transactions for verification; instead, the Merkle tree saved in the block facilitates this process. Miners receive a reward upon successful transaction verification, and the validated transactions are permanently recorded

in the blockchain.

Consensus Algorithm: Mining is the procedure used to add new blocks to the blockchain and validate the existing ones. This process includes verifying transactions within a newly created block, a crucial step within the broader context of the consensus algorithm. Each transaction is uniquely identified by its cryptographic hash, derived from the corresponding transaction data stored in the block. The PoW is the first consensus algorithm employed in blockchain, notably adopted by Bitcoin as its foundational protocol. Under this protocol, participants must use their computational resources to solve a challenging cryptographic problem called the nonce discovery process. In contrast, Ethereum officially switched to the Proof-of-Stake (PoS) consensus algorithm in 2022 [24], aimed at reducing energy consumption and providing a resilient platform for novel scaling solutions. Notably, blockchain networks based on PoW or PoS are susceptible to selfish mining attacks, although this paper specifically focuses on the Bitcoin network. We focus on the Bitcoin network, which is the most well-known blockchain network. Given that PoW serves as the basis for numerous blockchain-based systems, Bitcoin's prominence makes it an ideal case study for analyzing the efficacy and security of PoW. By concentrating on Bitcoin, we aim to provide insights broadly applicable to blockchain systems and their security considerations.

Fork: In a real-world network, the time required for information to traverse from one node to another can be notably prolonged. Given the geographically dispersed nature of blockchain miners worldwide, it is feasible that certain miners operate within network environments characterized by substantial latency [34]. In the context of a blockchain network, scenarios may arise where two miners generate and propagate their blocks in close temporal proximity. Consequently, other network participants can receive these blocks in dissimilar sequences, prompting miners to prioritize validating the first received block [34]. In this particular scenario, inconsistencies arise, leading to the formation of a fork within the blockchain. This fork, induced by block propagation delays, is referred to as a *natural fork* and is an inherent phenomenon within an imperfect blockchain network [34]. As shown in Figure 5.2, Miner 1 and Miner 2 each independently mine a new block, referred to as $B1$ and $B2$, respectively. It is assumed that these blocks are generated and distributed almost simultaneously. At a specific time, some miners receive and validate $B1$ as the first block, while others validate $B2$ instead. As a result, this occurrence leads to the emergence of a natural fork in the blockchain, manifesting as two separate branches denoted as $B1$ and $B2$.

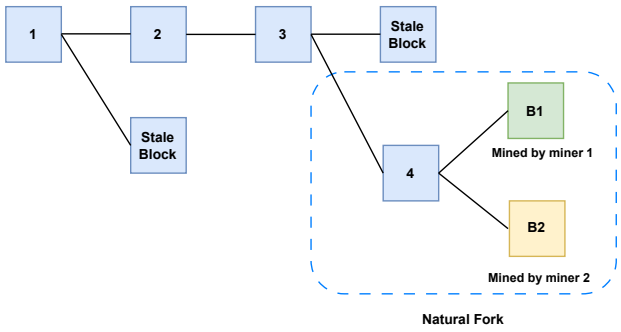


Figure 5.2 Example of a natural fork.

5.2.2 Investigation of Selfish Mining Attacks

Selfish mining is a disruptive strategy in blockchain systems based on PoW, which involves malicious miners or mining pools that do not immediately broadcast the newly mined block but choose to withhold blocks and release them later [54]. This creates a block race between honest and selfish miners. Selfish mining attacks can invalidate the blocks mined by honest miners. Thus, all transactions in the honest miners' blocks get rejected.

Figure 5.3 demonstrates three example states, where red blocks (blocks A and C in state 1, block A in states 2 and 3) represent mining by selfish nodes and gray blocks (the first two blocks of the chain, block B in all states, and block C in states 2 and 3) represent mining by honest nodes. The selfish mining attack can be successfully launched if the attacker is capable of mining at least one block ahead of honest miners. State 1 depicts a successful attack, while states 2 and 3 show two examples of unsuccessful attacks. In State 1, the attacker mines block C immediately after block A, causing the block B mined by the honest miner to become stale and resulting in a waste of computational resources for the honest miner. In contrast, in State 2, the attack is unsuccessful if the honest miner manages to mine a block after a block mined by the selfish miner. Block B becomes stale in this case, but the attacker does not lose any rewards. Furthermore, in State 3, the honest miner mines block C right after block B, which was mined by another honest miner, causing the selfish miner to lose.

For analyzing selfish mining attacks, we gathered several critical indicators from the literature review [54, 137, 180, 181]. Some of them are explained as follows:

- *Block receives time rate:* is the average time a Bitcoin node gets a new block. It measures the time that passes between a miner creating a block and the block being successfully received and appended to a node's copy of the blockchain. As soon as a new block is mined, it has to be broadcast throughout the network to notify all the

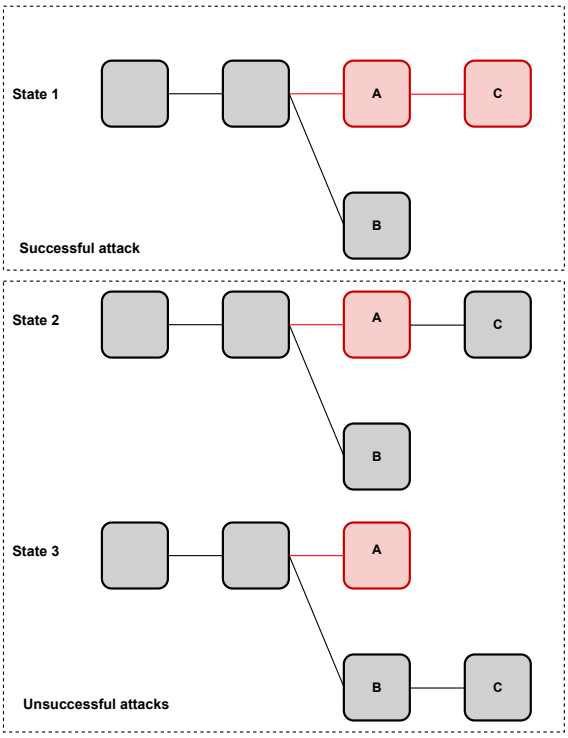


Figure 5.3 Selfish mining attack.

nodes in order to update their copies of the ledger. This metric can offer insights into the efficiency of the Bitcoin network's communication protocols. A consistently higher Mean Block Receive Time may indicate network congestion or communication issues, whereas a lower Mean Block Receive Time suggests efficient block propagation across the network.

- *Block propagation time rate:* the average travel time of a freshly mined block from its origin node to other nodes on the network where it is received and validated. The propagation time can fluctuate based on factors such as node connections, network latency, and protocol efficiency. A swifter propagation rate signifies improved network health and diminishes the probability of forks or conflicts within the blockchain.

- *Average block size*: It can be influenced by the number of transactions in the block. In certain instances, a selfish miner may opt to include fewer transactions in their block to expedite the mining of the next one [137].

- *Stale Blocks:* is known as an orphaned block, refers to a block successfully mined by a miner but not ultimately included in the main blockchain. In fact, after a fork, the stale block is excluded from the chain.

- *Miner Average Block Generate Interval*: measures the average time miners require to generate a new block, reflecting the mining rate and block production speed. It plays a critical role in assessing blockchain security and its commitment to the protocol's desired block generation time, such as Bitcoin's 10-minute goal.

- *Hash Rate:* refers to the computational power that a network or miner possesses. It quantifies the number of hashes a miner or network can perform per second.

- *INV received and sent message time:* relates to the timestamp associated with the reception of *INV* messages in a blockchain. The *INV* message, short for inventory message, transmits one or more inventories of objects known to the sending peer. It can be dispatched without a prior request to announce new transactions or blocks. The timing of *INV* messages is very important for analyzing selfish mining attacks. Selfish miners may hold the propagation of their mined blocks to gain a head start on honest ones. They strategically choose when to broadcast *INV* messages to the network to maximize their advantage.

- *GET_HEADERS received and sent message time:* The former is the timestamp when nodes receive *GET_HEADERS* messages, assessing network latency and potential selfish miner strategies. The latter metric measures the timestamp at which nodes send *GET_HEADERS* messages, which can affect their efficiency in block validation and mining. Both metrics are valuable for assessing selfish mining strategies and information propagation.

- *HEADERS received and sent message time:* play key metrics in selfish mining analysis. *HEADERS* received message time measures the delay experienced by honest miners when receiving block header information, potentially revealing if selfish miners are taking advantage of these delays. In contrast, *HEADERS* send message time tracks the timing of requests of nodes for block headers, influencing their effectiveness in block validation and mining. These metrics assist in evaluating how selfish miners strategically time their actions to gain a competitive edge in a selfish mining attack.

- *GET_DATA received and sent message time: GET_DATA* received message time denotes the timestamp when nodes receive *GET_DATA* messages, indicating the delay in honest miners accessing specific data, potentially revealing selfish miners' strategies. Conversely, *GET_DATA* sent message time records when nodes transmit requests for specific data, influencing their efficiency in obtaining essential information. These metrics contribute to the evaluation of whether selfish miners strategically time their data requests for an advantage during selfish mining.

- *BLOCK received and sent message time:* calculates the timing of block reception and transmission, serving as a metric that can reveal potential strategies employed by selfish miners in selfish mining attacks.
- *BLOCK_IN_FORK:* counts the number of blocks within competing blockchain forks. Selfish miners often take advantage of hidden forks with more blocks than the public chain and strategically reveal them to disrupt consensus. Tracking this metric can be helpful in selfish mining attack detection.
- *25% percentile of block propagation time:* The 25th percentile of block propagation time, which is commonly known as the first quartile in statistical analysis, signifies the point below which 25% of the recorded block propagation times fall. A lower 25th percentile indicates that a significant portion of the block propagation times are relatively fast, indicating good performance in terms of how quickly blocks are transmitted and validated in the blockchain network.

Existing research has explored additional indicators, like fork height. However, determining the fork height poses a challenge because only nodes situated at the edges of the fork can provide information about its height. If these nodes happen to be malicious, it becomes impossible to accurately ascertain the exact height [137]. Stale blocks are also considered significant indicators of selfish mining. However, accurately quantifying the number of orphaned blocks in the real world proves challenging since natural forks may occur, and distinguishing between orphaned blocks belonging to natural forks and those related to selfish mining is a complex task [137].

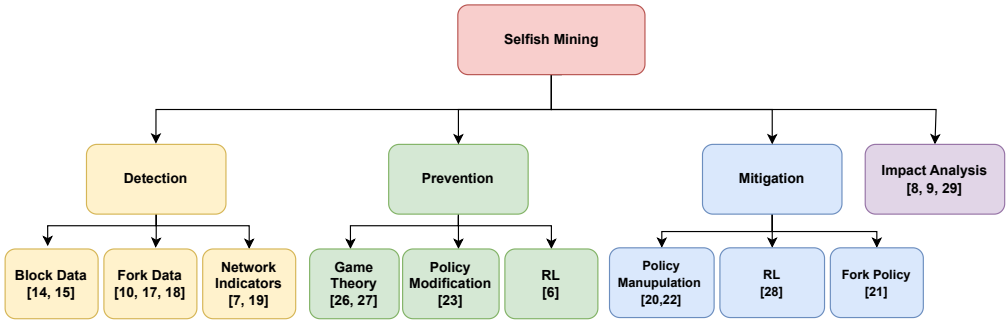


Figure 5.4 Taxonomy of existing defense approaches.

5.3 Related Work and Defense Taxonomy

The blockchain's unique features expose vulnerabilities that attackers can exploit. Any approach that provides attackers with control over the majority of the hashing power can severely compromise the integrity of the blockchain. When an attacker controls a significant portion of the hash rate, they reach the potential to execute other attacks, like a double-spending attack, by trying to replace the existing transaction history with an alternate one, creating a fork in the blockchain. In traditional network control scenarios, attaining a majority share of the hashing power is regarded as the critical threshold. Nevertheless, the advent of selfish mining strategies in PoW systems has revolutionized this perspective. Although selfish mining attacks have become more prevalent, limited research is currently dedicated to their detection and mitigation. Eyal and Sirer's pioneering work [54] introduced the concept of selfish mining, enabling attackers to increase their payoffs with only 25% of the total mining power - a stark contrast from the traditional 51% threshold. Expanding upon this innovation, Sapirshtein et al. [134] and Bai [135] developed the concept of optimal selfish mining, pushing the threshold even lower to a remarkable 23.21% and 21.48%, respectively.

As shown in Figure 5.4, existing works focusing on the security of blockchain against selfish mining attacks can be categorized into four categories according to the defense aspect, namely detection, mitigation, prevention, and impact analysis techniques. Existing works focus on three subcategories to detect the attack such as data related to natural forks, network indicators, and data related to blocks or transactions. Other works use fork policy, reinforcement learning (RL), and policy manipulation techniques to mitigate selfish mining attacks. For prevention, the existing works leverage some techniques such as game theory, policy modification, and RL. For impact analysis, prior studies concentrate on assessing the influence of this attack on the network and exploring potential factors contributing to selfish mining attacks.

Recently, Wang et al. introduced ForkDec [136], a fully connected neural network for selfish mining attack detection. They use a simulated dataset of natural and selfish forks in a blockchain network. ForkDec demonstrated superior performance in comparison to traditional selfish mining detection methods. More precisely, when utilizing artificial neural networks (ANN), the detection accuracy reached 99.03%. Chicarino et al. [138] presented a simple technique for recognizing selfish mining attacks in blockchain systems by monitoring the height of blockchain forks and determining a potential attack when the deviation from the expected value is substantial, typically exceeding a threshold of 2. Although this technique seems to be an effective approach as its accuracy is 98.49%, the likelihood of false positives or mistakenly identifying a natural fork as a selfish fork increases.

Peterson et al. [137] analyze selfish mining attacks by examining block-related data through the SimBlock simulator. However, their study lacks an implementation component and primarily focuses on reviewing certain features. Saad et al. [66] utilized transaction confirmation and block publishing heights to detect selfish mining in PoW-based blockchains. They established a “truth state" for each block to distinguish between honest and selfish blocks. Li et al. [139] proposed a statistical test for analyzing miner behavior in cryptocurrencies like Bitcoin, Litecoin, Ethereum, etc. By identifying anomalies in miners' successive block discoveries and employing heuristics-based address clustering, the study detects instances of unusual mining behavior in Monacoin and Bitcoin Cash but fewer instances in Ethereum.

Gobel et al. [182] extend the selfish mining model regarding network indicators. Using the spatial Poisson Process, they explore how propagation delay impacts the blockchain. The findings reveal that in the absence of any miner following the selfish mining strategy, there tends to be a substantial number of blocks in the blockchain, resulting in higher mining rewards for all miners. However, this work does not delve into the specific distribution among multiple selfish miners. On the other hand, the proposed approach in [179] focuses on enhancing the security analysis of selfish mining by considering both propagation delay and the distribution of selfish miners. In contrast to [182], which treated selfish miners as a unified group, this study aims to provide insights into individual selfish miners' specific dynamics and behaviors.

Heilman et al. [140] proposed a technique where miners are supposed to add unforgeable timestamps to their blocks. This makes the blocks created by malicious users worthless and prevents them from being added to the blockchain by selecting blocks with the most recent timestamps. However, one drawback is that there is a need for generating tamper-proof timestamps. Also, honest miners must maintain records of recent timestamp release logs. Zhang and Preneel [141] introduced a fork-resolving policy for handling blockchain conflicts by selectively ignoring blocks that are delayed in publication while favoring blocks that reference competing predecessors. When a secretly mined block is not made public until a competing block emerges, it does not contribute to either branch and loses its competitive advantage. Lee et al. [146] proposed a strategy involving transaction creation time to counteract selfish mining by selecting the most suitable block when multiple blocks emerge concurrently. This approach minimizes the probability of an honest mining pool generating a selfish block. However, a significant drawback is the requirement for all nodes in the network to achieve synchronization to guarantee simultaneous possession of identical information.

Reno and Sultana [147] introduce an efficient selfish mining prevention mechanism centered around block timestamps and the introduction of Alarming Blocks. Notably, this approach

avoids additional overhead on miners and effectively eliminates intentional forking within the network. However, a drawback is its limited applicability,which is primarily suited for a public blockchain using PoW. Moreover, the introduction of Alarming Blocks, which lack any transactions, may raise concerns about the efficiency and utility of the blockchain. Blockchain networks often prioritize transaction throughput and utility, and the inclusion of blocks with no transactions could be seen as a limitation in terms of resource optimization. Utilizing game theory [183–186] and reinforcement learning [144, 145] can also assist in reducing the risk of selfish mining by anticipating unusual behavior and adjusting the defensive strategies of fellow miners.

Yang et al. [34] quantitatively evaluate selfish mining's influence on blockchain networks, considering factors like block propagation delays and uncle-nephew block distances in Ethereum. Using the Markov model, they evaluated mining revenue, system performance, and other security metrics. Kang et al. [152] expand the analysis of selfish mining effects in Bitcoin and Ethereum networks with natural and extended forks. They emphasize that the revenue ratio of a selfish mining pool increases when forks occur more frequently [153].

A comprehensive overview of the research on selfish mining attacks through a blockchain network was described in this section. The existing work can be classified into four main groups: prevention, mitigation, detection, and impact analysis approaches. Some works use machine learning or data analysis techniques to detect this attack. Others provide mitigation approaches like timestamping or fork-resolving tactics. In addition, some prevention strategies can reduce the risk of the mentioned attack as game theory approaches, while impact analysis studies evaluate the total impact of these assaults on blockchain networks.

5.4 Proposed Methodology

In the first step, we analyze the state-of-the-art ForkDec technique [136], which uses a fully connected neural network with 12 layers to detect selfish mining attacks. We aim to use a random forest classification that yields lower computation overhead using the same dataset. We leverage the PCA method to reduce the dimensions of features. Then, we train an RFC model on the dataset of ForkDec for comparison purposes and to highlight the efficiency of our approach. In the second step, we generate our own dataset and use PCA to prepare the dataset for training an RFC model. Then, we compare the results with existing work. This section explains the system model, threat model, and methodology steps in detail.

5.4.1 System Model

We assume a Bitcoin network containing null nodes and miners. Null nodes do not store or validate the entire blockchain but only store and maintain a subset of the blockchain. Miners are entities participating in the blockchain that mine the blocks in the network. Two types of miners attend the blockchain: 1) *Honest* miners that follow the PoW algorithm and mine blocks according to the rules, and 2) *Selfish* miners that intend to deviate from the PoW algorithm and follow their own strategy.

5.4.2 Threat Model

Our analysis defines a threat model to encompass the potential risks and vulnerabilities associated with the selfish mining attacks within a Bitcoin network. Drawing inspiration from prior research and analysis, we define our threat model based on the strength and impact of the selfish mining attack. Our model considers the scenario where a single malicious miner employs selfish mining as the primary attack strategy. This choice is motivated by the understanding that the most potent form of selfish mining typically arises from a single attacker who is more profitable than multiple attackers and inflicts the most severe damage to fairness [135, 150]. We can also assume one single attacker as a single malicious pool in an environment of two pools, one honest and the other malicious.

The selfish mining attack, depicted in Figure 5.3, involves a series of strategic [54] steps initiated by a malicious miner. In the beginning, the attacker creates a private chain, keeping it concealed from the honest miners. Upon successfully mining a new block ahead of the honest miners, this block is incorporated into their private chain. Subsequently, the attacker continues to mine additional blocks on their private chain. Unaware of the newly mined blocks, honest miners persist in their mining operations and publish their delayed blocks. When honest miners eventually broadcast their newly mined blocks, the selfish miner rapidly disseminates their previously concealed blocks, triggering a competitive scenario. Some honest miners may accept the blocks from the selfish miner, while others opt for the blocks mined by their honest counterparts. Due to the longer private chain created by the selfish miner, their blocks have an elevated probability of acceptance over the blocks produced by honest miners. This strategy involves strategically withholding mined blocks to create a longer chain, thereby increasing the chances of the selfish miner's chain being accepted by the network.

This scenario reflects a situation where the selfish miner has managed to mine a single block ahead of others. However, if the selfish miner succeeds in mining two or more blocks ahead, they can reveal two secret blocks as soon as they receive a block from one of the honest miners.

As a result, the primary blockchain will undoubtedly shift to the selfish fork, recognizing it as the longest chain and guaranteeing that the selfish miner gains the rewards for these two blocks.

5.4.3 Our Proposed Detection Approach

First, we apply PCA to the dataset generated by [136] (referred to as “dataset 1" in this paper), as well as our generated dataset (referred to as “dataset 2") derived from a Bitcoin network using the Gervais simulator based on NS3 [180]. Next, we train our RFC model on both datasets. PCA analysis helps us examine their features and identify the most informative indicators by reducing the dimensionality of the features, thereby preparing the datasets for RFC. Following this preprocessing step, we train and test our model on these updated datasets with the objective of detecting selfish mining attacks in a Bitcoin network. The detailed steps are shown in Figure 5.5 and explained as follows.

Data Generation

First, dataset 1, containing training and testing data, is used to train and test the model. The dataset gathered from [136] includes 200,000 fork samples, with a ratio of 3 natural fork samples to every seven attacking fork samples. Additionally, we construct dataset 2 with 32 network features, including 102,400 entities. The simulator captures all network features from a benign network and a network under selfish mining attacks. Then, the feature selection part is triggered. The network indicators obtained from the Bitcoin network have been aggregated and can be found in our GitHub repository. Some of them are explained in Section 5.2.

Data Preprocessing and Feature Engineering

Dataset 1 comprises data containing four features, including block height, fork length, the block distance between this fork and the previous one, and the time difference between the timestamps of the first block of the fork and the previous one. We use PCA on dataset 1 to obtain a dataset with a single feature that can be effectively utilized for machine learning purposes without a substantial loss of information within the dataset. Hence, a PCA model with a single principal component is initialized. Next, we apply PCA to the training and testing data separately. The *fit_transform* method in the *scikit-learn* library is used to both fit the PCA model to the training data and transform it. For the testing data, the *transform* method is used to apply the transformation learned from the training data, ensuring consistency.

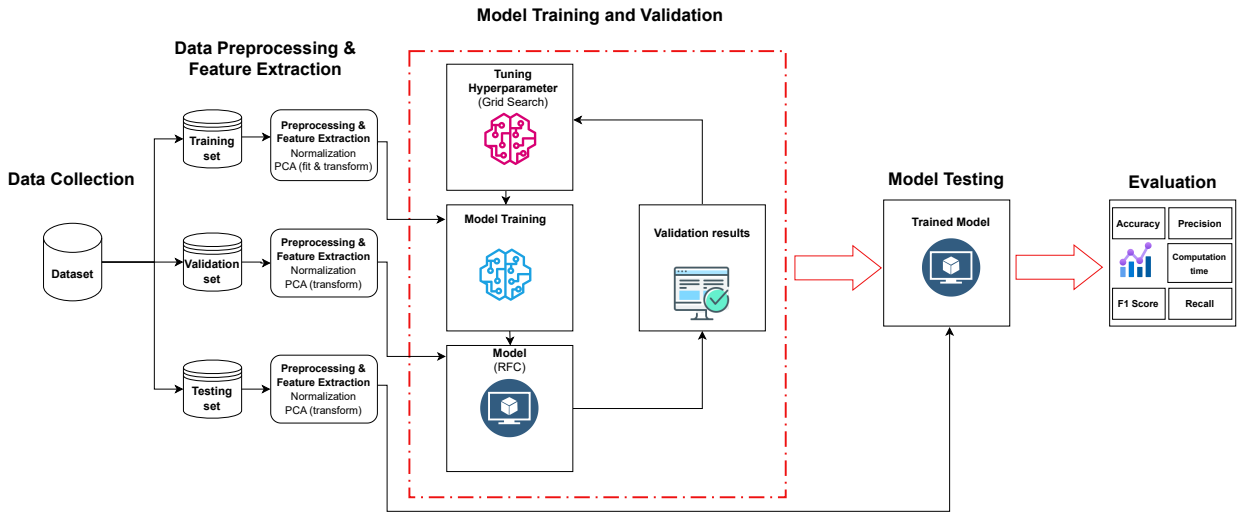


Figure 5.5 Our proposed approach for detection of selfish mining attacks.

Following a comprehensive analysis of all the features in the new dataset, we identified block receives time, block size, 25th percentile of block propagation time, 75th percentile of block propagation time, 90th percentile of block propagation time, and the average traffic per node are the most effective indicators for detecting selfish mining attacks. We use PCA on our dataset to gain insights into the variance explained by each component and the singular values associated with these components. This analysis assists us in making informed decisions about how many principal components to retain, facilitating dimensionality reduction and feature selection. Essentially, PCA helps assess each component's information content, enabling us to choose whether to retain all components or opt for a subset that simplifies our data while preserving its essential characteristics. The general steps of using PCA in our method are explained as follows.

1. Compute the Mean Vector:

$\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$

2. Compute the Covariance Matrix:

$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T$

3. Compute the Eigenvectors and Eigenvalues of $\mathbf{C}$:

$\mathbf{C}\mathbf{v}_i = \lambda_i\mathbf{v}_i$ for $i = 1, 2, \ldots, m$

4. Sort the Eigenvectors by decreasing Eigenvalues:

$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$

5. Select first k Eigenvectors to form the Projection Matrix:

$\mathbf{P} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$

6. Project the original data onto the new feature space:

$\mathbf{Y} = \mathbf{X}\mathbf{P}$

PCA transforms the original dataset $\mathbf{X}$ into a new set of variables by combining the original features in a linear manner. Here, X denotes the original data with dimensions $n \times m$, where m represents the mean vector, C is the covariance matrix, $\mathbf{v}_i$ are the eigenvectors, λ_i are the corresponding eigenvalues, k is the number of principal components to retain, P is the projection matrix, and Y is the transformed dataset in the reduced feature space. Initially, the mean vector and covariance matrix of the original dataset are calculated. Next, it determines the eigenvectors and eigenvalues of the covariance matrix, from largest to smallest. Then, PCA selects the top eigenvectors to construct a projection matrix and projects the original data onto a new feature space represented by the transformed dataset. This process reduces the dimensions of the data while preserving as much variance as possible, assisting in data analysis and visualization.

Data normalization was carried out using the Min-Max scaling technique to ensure the quality and compatibility of the input data for the classification task. By rescaling the feature values to a common range of [0, 1], we maintain the relative relationships between data points while addressing potential issues stemming from varying scales across features. To achieve this, we utilized the Min-Max scaling method implemented in the *scikit-learn* library. The resulting dataset, which underwent normalization, was kept for subsequent use in our classification analysis. This data preprocessing pipeline ensures that the features are uniformly scaled and suitable for various machine learning algorithms. This ultimately improves the performance and robustness of our classification model.

Model Selection

The authors in [136] use fully connected neural networks with 12 layers and 12 neurons. After analyzing several ML models in terms of the problem statement, computation, communication, and storage complexities, we select Random Forest Classification as it is well-suited for attack detection rather than a fully connected neural network. Hence, based on our problem, we need to train an RFC classifier to distinguish between benign and selfish networks.

Model Training and Hyperparameter Tuning

A rigorous training process was undertaken to construct an effective classification model for detecting selfish mining attacks. The dataset was first split into training and testing subsets using the *train_test_split* function from the *sci-kit-learn* library, with a test size of 20% and a fixed random seed for reproducibility. This partitioning ensured that the model's performance could be evaluated on an independent dataset. We adopt a comprehensive hyperparameter tuning strategy to optimize the hyperparameters of RFC and enhance its

performance. A hyperparameter grid (param_grid) is defined, encompassing various combinations of hyperparameter values, including the number of estimators (trees), maximum tree depth, minimum samples for splits, minimum samples for leaf nodes, and the maximum number of features considered during each split.

Grid search: The hyperparameter optimization was performed using the *GridSearchCV* function, which conducts a grid search over the defined parameter grid. This technique systematically explores different configurations by training and evaluating the classifier through cross-validation. Our approach utilized a 5-fold cross-validation strategy to ensure robust evaluation. Leveraging the computational efficiency of parallel processing, this systematic method identified the hyperparameters that yielded the most favorable classification results. The specific hyperparameter values tested during the grid search were as follows:

- *n_estimators* specifies the number of decision trees (estimators) in the Random Forest ensemble. We experiment with three different values: 50, 100, and 150 trees.
- *max_depth* governs the maximum depth of individual decision trees in the ensemble. We have specified three choices: None (allowing unlimited depth), 10, and 20. Setting it to None means that trees can grow until they contain fewer samples than the minimum required for a split (min_samples_split).
- *min_samples_split* establishes the minimum number of samples required to split an internal node. We test three values: 2, 5, and 10. It determines how finely the tree can split the data.
- *min_samples_leaf:* defines the minimum number of samples required to be at a leaf node (terminal node). We use three values: 1, 2, and 4. It can be helpful to decrease overfitting by setting a minimum size for leaf nodes.
- *max_features:* dictates the maximum number of features to consider during each split, introducing an element of randomness to the model. 'sqrt' and 'log2' typically imply the square root and base-2 logarithm of the total number of features, respectively.

This optimization results in a finely tuned Random Forest Classifier ready for deployment to detect selfish mining attacks with enhanced accuracy and robustness.

Testing the Model

Next, we evaluate our trained RFC model. Once the optimal hyperparameter configuration is determined, the model with these optimal settings is selected. Initially, a cross-validation

scheme was employed on the training dataset, employing a 5-fold cross-validation method. This process involves training the model on various subsets of the training data and assessing its performance regularly across different partitions. The classification accuracy is recorded for each fold, and the mean cross-validation accuracy is calculated to provide a robust estimate of the model's performance. Furthermore, we evaluate the model's performance on the training dataset by analyzing the number of misclassifications and calculating the training accuracy. This step assists in understanding how well the model fits the training data.

Performance Evaluation

We assess the performance of our model on a separate test dataset using several evaluation metrics like precision, recall, accuracy, and F1 score, which provide insights into the model's ability to classify instances of selfish mining attacks and benign mining correctly. This evaluation aims to comprehensively understand the RFC model's effectiveness in detecting selfish mining attacks and its generalization capabilities beyond the training dataset.

5.5 Experimental Results

Our experiment is designed to showcase the accuracy and efficacy of our method for detecting selfish mining attacks. We performed classification using RFC on an existing dataset [136] (dataset 1), containing four features relevant to selfish mining attack detection. After analyzing these features, we discovered that selecting only one feature using PCA can achieve higher accuracy with reduced computation, communication, and storage overhead. After splitting the dataset into training and test sets, we trained the RFC model on the training data to learn patterns and relationships between the features. Subsequently, we applied the trained model to the test data to make predictions. The machine used is Ubuntu 22.04 LTS, Core i5, 3.4GHz, with 16G RAM.

In the second step, we simulated a Bitcoin network using the Gervais simulator, based on NS-3 [180], and collected our dataset (dataset 2). Our dataset contains data from both benign and selfish Bitcoin networks, gathered via 100 iterations. We analyzed all 32 features and identified six independent features as the best indicators for selfish mining attacks within the simulator. These include block receive time, block size, 25th percentile of block propagation time, 75th percentile of block propagation time, 90th percentile of block propagation time, and traffic per node. We implemented our detection approach on the generated dataset and compared the obtained results.

Based on the results shown in Table 5.1, our approach outperforms existing work in terms

Table 5.1 Performance comparison

Evaluation metrics	Dataset 1		Dataset 2	
	(FC-NN)	(PCA + RFC)	(FC-NN)	(PCA + RFC)
Accuracy (%)	98.98	99.96	92.34	96.40
Recall (%)	1.0	99.76	99.07	99.08
Precision (%)	95.86	1.0	87.53	94.17
F1 Score (%)	97.88	99.88	92.94	96.56
Execution time (sec)	442.88	109.14	15.33	0.37

of accuracy, precision, recall, F1 score, and execution time. In this case, the accuracy of a fully connected neural network (FC-NN) model on both the existing dataset 1 and the new dataset 2 is lower compared to our proposed approach based on PCA and RFC. The FC-NN approach achieves an accuracy of 98.98% on dataset 1, whereas our approach already delivers an accuracy rate of 99.96% on the same dataset. Furthermore, when applied to dataset 2, FC-NN achieves an accuracy of 92.34%, while our approach yields an accuracy of 96.40%, demonstrating its superiority over existing work. Additionally, our approach significantly improves time complexity, reducing the time required for computation on dataset 1 to 109.14 seconds, whereas FC-NN requires 442.88 seconds. Also, on dataset 2, FC-NN takes 15.33 seconds, whereas our approach only requires 0.37 seconds, making it highly suitable for real-time attack detection in large blockchains.

In terms of computation and storage overhead, deep learning models are more complex than RFC. The ForkDec approach consists of 12 layers, which implies a greater need for communication. In contrast, Random Forest does not entail significant communication overhead during training because it is an ensemble of decision trees, with each tree trained independently. Additionally, FC-NN has a higher storage requirement due to model weights, activation, and gradients. In comparison, the structure of decision trees in RFC results in individual trees that are typically smaller in size than deep neural networks. Regarding computation, the 12 layers in deep learning models are computationally intensive. If we can achieve better results with the ML model, there is no need to employ a complex deep learning model, such as a fully connected neural network, especially in large-scale and resource-constrained networks.

5.6 Conclusion

Selfish mining attacks pose significant challenges to blockchain security, affecting networks such as Bitcoin, Ethereum, Monacoin, and other blockchains. This paper introduces a new method leveraging PCA and RFC to detect selfish mining attacks. Initially, we apply our model to an existing dataset and compare its performance with state-of-the-art techniques to highlight the efficiency of our approach in selfish mining detection. Furthermore, we delve into analyzing network indicators associated with selfish mining attacks. To achieve this,

we simulate a Bitcoin network and collect relevant network features. After confirming the practicality of capturing these features in a real-world Bitcoin network and ensuring their independence from simulator parameters, we select six critical features and create our dataset. Subsequently, we apply our proposed approach and compare it with existing methods. Our method achieves a significant accuracy rate of 99.96%, surpassing complex deep learning models while substantially reducing both computational and time complexity.

Our emphasis on optimizing the use of computational and storage resources further sets our work apart. Utilizing PCA and RFC alone, our approach not only enhances accuracy but also minimizes the reliance on complex deep learning models, showcasing a more efficient and pragmatic solution for selfish mining attack detection. In the future, we plan to enhance the existing simulator to make it more accessible for researchers to analyze network indicators related to other security attacks on blockchain. Additionally, we aspire to implement more practical solutions to mitigate selfish mining attacks across various blockchain networks.

CHAPTER 6 ARTICLE 3: RETALIATION GAME FOR MITIGATING SELFISH MINING ATTACKS IN BLOCKCHAIN NETWORKS

Fatemeh Erfan, Martine Bellaiche, Talal Halabi

Published on 13th EAI International Conference on Game Theory for Networks (EAI GameNets), Cambridge, UK, December 31, 2024

Abstract

Blockchain networks are exposed to several security attacks, including selfish mining attacks, which occur when a miner or a group of miners withhold their newly mined blocks and keep them in their private chain rather than immediately broadcasting them to the rest of the network. This strategy forces honest miners to waste their resources and degrades the efficiency of the network. This paper introduces a novel game-theoretic approach leveraging a war of attrition framework to mitigate selfish mining in PoW blockchains. We design reward and punishment mechanisms to incentivize miners to opt for honest strategies and discourage them from choosing selfish tactics. Additionally, a reputation function is utilized to ensure that miners are not indifferent to the presence of selfish miners in the network. Moreover, we examine whether the honest strategy remains stable within the system using the evolutionary game theory. Our findings indicate that the honest strategy is indeed stable under the proposed model. Our solution will help mitigate the threat of selfish mining in blockchain systems.

Keywords: Selfish mining attack, evolutionary game theory, retaliation game, threat mitigation, blockchain systems.

6.1 Introduction

Blockchain is a decentralized peer-to-peer (P2P) network that enables transactions to be verified by a group of untrusted participants. All participant transactions are recorded in a transparent public ledger known as the blockchain. Blockchain users adhere to a set of rules called the consensus protocol to add a new record known as a block to the chain, facilitating the agreement among most users on a unified record. Blockchain offers participants significant features such as integrity, availability, decentralization, immutability, fault tolerance, and transparency [15]. These characteristics have gained substantial attention from academia and industry in recent years.

The pioneering decentralized cryptocurrency, bitcoin, was introduced as a revolutionary con-

cept that leverages proof of work (PoW) as a consensus protocol [20]. In a Bitcoin network utilizing a PoW-based consensus algorithm, users do not need to authenticate to enter the network. This feature enhances the scalability of the Bitcoin consensus model, allowing it to support thousands of network nodes [187]. Among the myriad security threats blockchain technology faces are denial of service (DoS) and distributed denial of service (DDoS) attacks. Notably, selfish mining attacks [54] represent a form of DDoS threat designed to force honest miners to squander their resources, disrupting their ability to effectively access blockchain network services.

Selfish mining is an adversarial strategy in PoW blockchain systems where malicious miners or mining pools deliberately withhold newly mined blocks instead of broadcasting them immediately. By strategically releasing these blocks later, they initiate a race for block validation between honest and selfish miners. This tactic causes the blocks mined by honest participants to become orphaned, leading to the rejection of all associated transactions [54].

Multiple works focus on analyzing the behavior of participants in blockchain networks. Many techniques are leveraged to mitigate and detect selfish mining attacks, including game theory, machine learning, and analyzing network indicators such as fork rate and the number of orphan blocks. Game theory has found extensive application in modeling interactions and conflicts among participants. Many works have utilized various types of game theory to mitigate such security attacks. In this paper, we propose a game-theoretic approach to mitigate selfish mining attacks within a Bitcoin network by leveraging a type of retaliation game called a war of attrition.

Our contributions are as follows.

- We leverage a type of retaliation game named a war of attrition to mitigate selfish mining attacks. To the best of our knowledge, this is the first time this game has been used to mitigate selfish mining attacks.
- We introduce the punishment, the reward, and the reputation systems in order to penalize malicious actions and encourage honest actions within the blockchain network.
- We simulate and analyze our model by utilizing an evolutionary game to study the impact of whether the honest strategy is in a stable state.

The rest of the paper is as follows. Section 6.2 explains background concepts regarding blockchain security and game theory approaches. Section 6.3 describes existing work proposed to mitigate selfish mining attacks in blockchain systems. Section 6.4 presents the

proposed game framework. In Section 6.5, the proposed game will be analyzed. Section 6.6 demonstrates the results of the proposed game, and finally, Section 6.7 concludes the paper.

6.2 Background Concepts

This section presents background information related to some blockchain preliminary concepts and game theoretic approaches.

6.2.1 Blockchain

Block. Blockchain transactions are recorded in blocks, which miners sequentially link to form the blockchain starting with the genesis block. Each block has a header and a body: the body contains transaction data in a Merkle tree, where hashes combine to form the Merkle Root [22]. The header includes the block's identifier, previous block reference, timestamp, nonce, and Merkle root.

Miner. A miner generates new blocks in the blockchain. Miners solve cryptographic challenges to create new blocks using a consensus mechanism, typically PoW in Bitcoin. These blocks are then verified by the network, utilizing the Merkle tree structure for efficient validation. Upon successful validation, miners are rewarded, and the transactions are permanently recorded on the blockchain.

Consensus Algorithm. The mining process validates the blockchain by verifying transactions in each new block, which is central to the consensus algorithm. In Bitcoin's PoW, participants solve a cryptographic challenge (nonce discovery) using computational power. Ethereum switched to Proof-of-Stake (PoS) in 2022 to reduce energy use and support scaling [188]. Both PoW and PoS are vulnerable to selfish mining attacks, though this study focuses on the Bitcoin network.

Fork. A fork occurs when two miners create and broadcast separate blocks simultaneously. Miners then select the first block they received as the primary one, resulting in only one validated block. Forks can also be exploited for double-spending and selfish mining attacks [189].

6.2.2 Game Theory

Game theory has proven an effective tool for dealing with strategic challenges in blockchain networks, such as resource competition and decentralized decision-making like the Internet

of Things (IoT) [190]. Researchers have used game-theoretic frameworks to analyze security problems in blockchain-enabled networks and create strategic defenses against attackers [65]. Here, some essential concepts, game types, and their application in blockchain will be explained.

Nash Equilibrium (NE). Solution concepts provide a framework of rules for forecasting the outcome of a game by identifying the strategies players are likely to adopt [65]. Nash Equilibrium is one such concept, where a game reaches equilibrium if each player's chosen strategy is optimal, given the strategies of other players. In this state, no player is incentivized to change their strategy to increase their expected payoff while others stick to theirs. When NE is reached, every player adopts a strategy that maximizes their potential reward.

Retaliation Game. The goal of this game is to punish the selfish player. Here, the player replies to the opponent's strategy to punish them if they have an undesirable action and make them pay a cost. The responses can be mirroring their opponent's actions, setting some penalties, paying some cost to the opponent, or even taking a counterattack to neutralize their action. If we consider decreasing value over time, we normally use a war of attrition.

Tit-for-Tat. Tit-for-tat (TFT) is a retaliation strategy where a player mirrors the opponent's previous action. Initially, the player cooperates; if the opponent cooperates, they continue to do so. If the opponent behaves selfishly, the player mirrors it in the next move. This strategy is ineffective against selfish mining because honest miners lack the computational power to match an attacker's selfish behavior. In equilibrium, both players cooperate continuously if they both start by cooperating [191].

War of Attrition. The theory of war of Attrition introduced by Maynard Smith [105] analyzed a scenario where two animals are in competitive encounters over a single resource valued v. As the conflict endures, both parties experience increasing costs. In particular, if players choose to continue fighting, they accumulate costs until one of them concedes. Each player's costs escalate the longer they remain engaged in the encounter, adding pressure to resolve the conflict. Notably, the distinctive symmetric equilibrium manifests as a mixed strategy equilibrium, where participants exhibit decreasing probabilities of getting involved in each encounter. This equilibrium demonstrates dual characteristics, serving as an evolutionarily stable strategy (ESS) and a subgame perfect equilibrium (SPE) - a NE enhancement. If there is an imbalance between players - if one starts with a resource advantage, choosing not to fight can be an ESS [106].

Evolutionary Game. The study of evolutionary games examines how players with limited rationality can augment their outcomes over time through repeated interactions [110]. In this model, players who gain more revenues are more likely to continue participating in successive rounds, whereas those with lower payoffs tend to drop out. Through this iterative process, the system naturally favors strategies that yield better payoffs, leading to an equilibrium where dominant strategies stabilize across the population. Using evolutionary and retaliation game concepts, we propose a solid model that captures both the strategic, long-term dynamics of conflict escalation and the tactical, situational decision-making processes within individual encounters known as the War of Attrition.

6.3 Related Work

Some existing work focuses on mitigating selfish mining attacks in blockchain-based networks. A few papers study the application of game theory in blockchain-based networks to prevent, mitigate, and detect security attacks, including selfish mining attacks [97], [65]. Eyal [148] models a non-cooperative game to examine relationships between mining pools. In this scenario, a selfish pool can increase its utility by exploiting honest nodes and infiltrating miners who perform block withholding (BWH) attacks within the targeted pool. However, this strategy is effective only when one pool controls a substantial share of computational power; otherwise, mutual attacks reduce profitability.

Luu et al. [142] use the Computational Power Splitting (CPS) game to model pool block Withholding attacks. In this attack, miners or pools can boost their rewards by either redirecting computational power for BWH attacks or following the pool's protocol. When attacking, they allocate a portion of their computational power to maximize the attack's effectiveness. Zhen et al. [143] model mining between two miners as an iterative game and introduce a Zero Determinant (ZD) strategy to alleviate the miners' dilemma. This pinning strategy enables an altruistic miner to influence the selfish miner's payoff, encouraging cooperation and increasing overall social welfare.

Eyal et al. [54] indicated that an attacker must possess a computational power of at least $\frac{1-\gamma}{3-2\gamma}$ for selfish mining to yield a profit, where γ represents the fraction of blocks created by honest miners that the attacker successfully adopts or observes. Consequently, the attacker would require a minimum computing power of 0.09, even under optimal network conditions. Based on [149], in the case of Nash equilibrium on multiple, non-uniform bitcoin block withholding, the smallest pools might maximize their profits by not retaliating at all. In other words, retaliation is not considered a rational move for the smallest pools. In some existing works, the authors utilize the TFT strategy to mitigate any deviation from honest mining through

the blockchain network. The TFT strategy is a mutual strategy [150] in which the player chooses his strategy based on his opponent's strategy. The goal is to maximize your payoff. First, the player selects a cooperation strategy and watches the opponent. If the opponent decides to cooperate with player A, player A will choose to continue cooperation for the next move. If player B selects selfishly, player A mirrors the opponent's strategy.

After analyzing existing works on mitigating selfish mining attacks, we find that game-theoretic models like non-cooperative games, CPS, and ZD strategies provide insights into miner behavior and interactions. However, these approaches have limitations: they often rely on specific network conditions and computational power distribution, which can favor larger pools. Our research extends these findings by developing a more adaptive model that considers diverse network conditions, miner types, and resource allocation challenges.

6.4 The Proposed Game Framework

Prior to delving deeply into the dynamics of retaliatory strategies, we explain the model, players, objectives, actions, and potential states of selfish mining attacks. We consider a repeated game involving two players (malicious and honest) within a discrete time horizon. The individual game within this repeated game is (S_i, u_i), where $i \in \{1, 2\}$. Here, S_i denotes the set of all possible actions for player i, $s_i \in S_i$, $S \equiv S_1 \times S_2$, $s \equiv (s_1, s_2) \in S$, $u_i : S \to \mathbb{R}$, and $u_i(s)$ represents the payoff for player i resulting from action profile $s \in S$. The notations are shown in Table 6.1.

Players. Our game assumes two types of miners: selfish and honest miners. The selfish miner is incentivized to maximize their payoff by deviating from the consensus protocol. The honest miner aims to maximize their payoff and protect the integrity of the blockchain.

Strategies. Players have two options to choose from and build their strategy: mining honestly or maliciously. The attacker intends to conceal his newly mined block, whereas the honest miner reveals his block (avoids concealing it).

Actions. The players have the following options:

- **Concealing:** the miner opts to keep their newly mined block and avoid broadcasting it.
- **Revealing:** The miner tends to publish their block and propagate it through the network.
- **Retaliating**: the player intends to respond to their opponent's actions in the previous

Table 6.1 Notations used in our model

Notation	Definition
P_0	The initial punishment value
$P(t)$	Punishment system
$\phi(t)$	Decreasing function for the reward system
$\psi(t)$	Increasing function for the punishment system
$R(t)$	Reward function: $R(t) = \phi(t) \cdot r$
C_m	The net cost of mining for the miner m
r	The value of a successfully mined block
rep	The reputation score representing miner contribution

round. The miner (assumed honest) aims to rent sufficient hash power from other honest miners to counter the attacker. In this scenario, the interaction between the attacker and the victim resembles a War of Attrition, where both players compete for a fixed prize by taking turns attacking each other. However, each attack incurs an additional cost for the attacker. Our findings indicate that, over time, the profit from each new attack diminishes for at least one player. Eventually, it becomes unwise for a player to continue counterattacking, leading to the end of the game, as it becomes more detrimental for a player to keep fighting than to stop. This action will be part of a concealing strategy, where the defender aims to retaliate by concealing their block in the next round. Therefore, we focus on concealing and revealing as the main actions in this game.

There are cases where a player might choose to retaliate against his opponent:

- *State transitions:* We can model the behavior of miners across the network using state transitions, where each state represents a particular configuration of actions and outcomes. The progression between states can be captured through the Markov process [54, 150]. Choosing the retaliation strategy becomes an option when one party's actions prompt a shift to a new state, requiring a strategic response. For instance, if the selfish miner conceals a block in State S_0, the honest miners might retaliate by enforcing a fork when informed of the concealed block.

- *Revealing blocks:* If the Selfish Miner conceals blocks to gain an advantage, the honest miners might retaliate by revealing their own blocks or enforcing forks to maintain the integrity of the blockchain and reduce the selfish miner's potential rewards.

- *Fork Competition:* When both parties compete to secure the next block, retaliation strategies can be employed as each side strives to gain dominance. These dynamics,

which might occur in states such as S'_0, motivate each miner to escalate their efforts to outperform the other.

- *Lead in blocks:* When the Selfish Miner maintains a lead in blocks (e.g., in State S_2 or S_n), the honest miners might retaliate by strategically revealing blocks or attempting to catch up in mining efforts to reduce the lead and disrupt the Selfish Miner's dominance.

Reward function. It is designed to decrease over time to encourage the miners to promptly broadcast their newly mined blocks. We define it as:

$$R(t) = \phi(t) \cdot r \tag{6.1}$$

where $\phi(t)$ is a decreasing function and r is the base reward for successfully mining a block. By making $\phi(t)$ decreasing over time, miners who delay their broadcasts receive a reduced reward, while those who broadcast immediately maximize their gains. This time-sensitive reward structure aligns miner incentives with the network's goal of maintaining timely and transparent block updates. In our model, t represents the combined delay defined as follows:

$$t = t_m + \delta \tag{6.2}$$

where t_m is the mining time and δ is the block propagation delay caused by network latency. This delay influences the reward and punishment mechanisms. Longer propagation delays increase the punishment and decrease the reward, discouraging miners from withholding blocks.

Punishment function. Prior research often assumes a fixed value for punishment. In contrast, this paper introduces a dynamic punishment model that increases over time. The punishment function should reflect the delay in broadcasting a newly mined block, thereby incentivizing timely behavior among miners. We define the punishment at time t as an increasing function, denoted by $\psi(t)$, which scales the initial punishment P_0 based on the time elapsed since the block was mined but not broadcast.

$$P(t) = \psi(t) \cdot P_0 \tag{6.3}$$

Here, P_0 represents the initial punishment imposed at the moment a miner fails to broadcast their block (i.e., at $t = 0$), and $\psi(t)$ is a function that grows with time t, representing an increase in punishment the longer the delay persists. To mirror the effect of reward attenuation, we propose that the punishment function $\psi(t)$ could take an inverse form of the component of the reward function $\phi(t)$. This approach implies a more significant penalty for

prolonged concealment. Consequently, we define:

$$P(t) = \phi(t)^{-2} \cdot P_0 \tag{6.4}$$

where $\phi(t)$ is the component of the reward function $R(t)$. Using $\phi(t)^{-2}$ in this manner ensures that as $\phi(t)$ decreases, the punishment $P(t)$ correspondingly increases, thus creating a disincentive for prolonged concealment.

Reputation score. If a player conceals, they receive a negative reputation score ($-rep$), while if they reveal the block, they gain a positive reputation score ($+rep$). This scoring system can enhance miners' incentives to choose revealing actions over concealing. Therefore, we assume that each node possesses its reputation matrix, updated after each block is published on the ledger. Upon request from other nodes, this matrix will be shared with them. This system is utilized to verify the blocks broadcast through the network. Hence, the reputation score of the publisher influences whether their block is accepted or rejected.

Given the above definitions, the following inequality holds:

$$|\psi(t)| > |\phi(t) \cdot r - C| > |rep| \tag{6.5}$$

to discourage selfish strategies among miners - the dynamic punishment grows large enough over time to outweigh the rewards for concealing, while reputation penalties remain impactful but secondary to the immediate financial incentives.

Assumptions. We consider some assumptions to render the payoff matrix dynamic and responsive to player strategies.

Assumption 1: Free entry logic derived from [192] states that in a blockchain network, anyone can freely join and contribute his computational power to the network; the equilibrium level of computational power allocated to blockchain mining denoted as n is determined by $n = R/C_m$, where C_m and R are considered the cost of mining and the reward of mining a block, respectively. If there are a few miners in the network ($n < R/C_m$), the reward of mining a block is higher than the cost of each miner. So, mining in this network (pool) is profitable. If there are too many miners in this pool, it means that the reward is less than the cost that a miner should tolerate.

Assumption 2: If the net payoff turns negative for any player, this signals the end of the game.

Assumption 3: The net cost of the selfish mining attack, c, remains constant (or changes

at a slower rate than $\phi(t)$).

Assumption 4: Both miners should pay a cost of C for the mining process. These costs are C_a and C_d for the attacker and the defender, respectively.

Assumption 5: If the attacker reveals his block, he gets $\phi(t) \cdot r$ as a reward, where in time 0 this reward is r.

Assumption 6: If the defender retaliates (chooses to conceal his block), he should pay a cost of P as punishment as well as a C_d for their mining efforts.

Assumption 7: If one conceals while another reveals, the player who reveals the block is not rewarded. This is because honest miners should cooperate to have a healthy network. If both reveal their blocks, the one who broadcasts first receives the reward, while the other does not, and their block becomes orphaned.

6.5 Equilibrium Analysis

$C_a, C_d > 0$ are considered the cost of a mining process for the attacker and the defender, respectively, and $r > 0$ is the reward for mining a block through the blockchain network. The malicious miner (M) chooses to reveal or conceal. We consider a function of time $\phi(t)$ to prove that the net cost of the attack falls over time, with $\phi(t) = 1$ when $t = 0$. We consider t as the period to mine a block.

6.5.1 Time to stop

The strategic aspects of this model involve optimizing the timing of stopping the retaliation for the miner, considering the costs of attack or mining C, the decreasing value of the reward $\phi(t) \cdot r$, and the potential reputation loss for the defender rep if they lose a round.

We consider that the player (honest miner) has two options: revealing (immediately broadcasting their mined blocks) and concealing (retaliating by withholding their blocks). The honest miner aims to determine the point, denoted as T_{stop}, at which it becomes unprofitable to continue retaliating against the selfish miner. We consider the cumulative revenue from the revealing and concealing actions to calculate T_{stop}.

If the revenue from choosing to reveal up to time t is represented by the cumulative sum:

$$R_{total}(t) = \sum_{k=0}^{t} \left(\phi(k) \cdot r - C_a + rep\right), \tag{6.6}$$

and the cumulative loss from concealing up to time t is given by the value:

$$L_{total}(t) = \left| \sum_{k=0}^{t} (-C_a - P - rep) \right|, \tag{6.7}$$

then, the stopping condition arises when the total revenue from retaliating equals the absolute cumulative cost of concealing. Thus, we find T_{stop} by solving:

$$R_{total}(T_{stop}) = L_{total}(T_{stop}), \tag{6.8}$$

or equivalently,

$$\sum_{k=0}^{T_{stop}} (\phi(k) \cdot r - C_a + rep) = \left| \sum_{k=0}^{T_{stop}} (-C_a - P - rep) \right|. \tag{6.9}$$

This condition indicates that retaliation should stop when the total accumulated revenue from revealing no longer exceeds the absolute cumulative losses incurred by concealing.

If a player chooses to conceal a block, his payoff is $R = -C_a - P - rep$ as P is the punishment function that is explained in 6.4. If he chooses to reveal the block, his payoff is $\phi(t) \cdot r - C_a + rep$ as rep is the reputation score explained in 6.4. It is worth mentioning that C_d and C_a depend on the cost of mining, the cost of renting computation resources, and the fee of joining any pool (if they are mining in a group).

6.5.2 Strategies

As shown in Table 6.2, this game has two primary players: the attacker and the defender. Each player can reveal or conceal the newly mined block. When a player conceals, they face a penalty, represented by P, and their reputation score will decrease, which refers to a financial deduction affecting their reputation.

If the attacker chooses to reveal the block, they stand to gain $\phi(t) \cdot r - C_a + rep$, where $\phi(t)$ represents a function of decreasing value over time t. This function reflects that the longer it takes to complete an action, the less the reward r becomes. C_a represents the cost of mining for the attacker. rep is the reputation score indicating the miner effectively contributes to

Table 6.2 Payoff Matrix for the Selfish Mining Attack

Attacker \ Defender	**Reveal**	**Conceal**
Reveal	$(\phi(t) \cdot r - C_a + rep, \phi(t) \cdot r - C_d + rep)$	$(-C_a + rep, -C_d - P - rep)$
Conceal	$(-C_a - P - rep, -C_d + rep)$	$(-C_a - P - rep, -C_d - P - rep)$

the pool (or network). Similarly, if the defender chooses to reveal a block, they can gain $\phi(t) \cdot r - C_d + rep$, where C_d is the cost of mining for the defender.

The strategies can be divided into two groups:

- Revealing: If the attacker reveals the block, they gain $\phi(t) \cdot r - C_a + rep$. Likewise, if the defender reveals the block, they gain $\phi(t) \cdot r - C_d + rep$.
- Concealing: If the attacker conceals the block, they gain $-C_a - P - rep$, where P represents the punishment. Similarly, if the defender conceals the block, they gain $-C_d - P - rep$.

It is worth mentioning that if both players choose to reveal their blocks, a competition will happen. The details of the actions are as follows:

- **Reveal (R) vs. Reveal (R):** Neither the attacker nor the defender has the incentive to change their strategy if both choose to reveal as the following inequation holds:

$$\phi(t) \cdot r - C_a + rep \geq -C_a - P - rep \tag{6.10}$$

- **Reveal (R) vs. Conceal (C):** If the attacker reveals while the defender conceals, the attacker could also benefit by concealing, given that the punishment for concealment decreases as time passes. The defender has no incentive to deviate. Thus, there is no Nash equilibrium in this cell.
- **Conceal (C) vs. Reveal (R):** If the attacker conceals while the defender reveals, the attacker could benefit by revealing, as the punishment for concealment increases over time. The defender has no incentive to deviate. Thus, no Nash equilibrium in this cell.
- **Conceal (C) vs. Conceal (C):**

  If both players choose to conceal the block, then the payoff is less than the revealing option. This scenario would not lead to a Nash equilibrium because both players would be incentivized to deviate from choosing to conceal.

Our model assumes perfect information, but practical detection of concealed blocks is achievable using network monitoring and anomaly detection techniques. Methods like analyzing fork rates, orphan block patterns, and machine learning can help miners detect selfish mining [193].

6.5.3 Using Evolutionary Game

The game between miners is assumed to be dynamic to make it more realistic. As the players' strategies evolve over time, leveraging *evolutionary game* becomes a suitable approach based on their received payoff. This game illustrates how strategies that perform better will become more prevalent in the population of attackers and honest miners. Evolutionary game theory helps identify stable strategy profiles, known as ESS. This game demonstrates whether a particular mix of revealing and concealing strategies will persist in the long run or if the system will evolve toward a different equilibrium.

Let x be the proportion of attackers using the Reveal strategy and y the proportion of defenders using the Conceal strategy. The expected payoffs for the attacker and defender using the Reveal and Conceal strategies are as follows.

Attacker revealing:

$$\pi_A^R = y \cdot (\phi(t) \cdot r - C_a + rep) + (1 - y) \cdot (-C_a + rep) \tag{6.11}$$

Attacker concealing:

$$\pi_A^C = y \cdot (-C_a - P_a - rep) + (1 - y) \cdot (-C_a - P_a - rep) \tag{6.12}$$

Defender revealing:

$$\pi_D^R = x \cdot (\phi(t) \cdot r - C_d + rep) + (1 - x) \cdot (-C_d + rep) \tag{6.13}$$

Defender concealing:

$$\pi_D^C = x \cdot (-C_d - P_d - rep) + (1 - x) \cdot (-C_d - P_d - rep) \tag{6.14}$$

The replicator dynamics show how the proportions of each strategy evolve based on the payoff differences. For the attacker:

$$\frac{dx}{dt} = x \cdot (\pi_A^R - \overline{\pi_A}) \tag{6.15}$$

where the average payoff for the attacker, $\overline{\pi_A}$, is given by:

$$\overline{\pi_A} = x \cdot \pi_A^R + (1 - x) \cdot \pi_A^C \tag{6.16}$$

For the defender:

$$\frac{dy}{dt} = y \cdot (\pi_D^R - \overline{\pi_D}) \tag{6.17}$$

where the average payoff for the miner, $\overline{\pi_D}$, is given by:

$$\overline{\pi_D} = y \cdot \pi_D^R + (1 - y) \cdot \pi_D^C \tag{6.18}$$

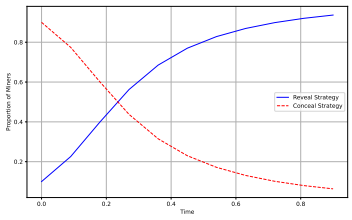
This analysis demonstrates how miners strategically choose to reveal or conceal blocks based on rewards, costs, and reputation. Honest miners can minimize cumulative losses by identifying the optimal stopping time T_{stop}. Evolutionary game dynamics further reveal the stability of these strategies in a competitive mining environment.

6.6 Numerical Results

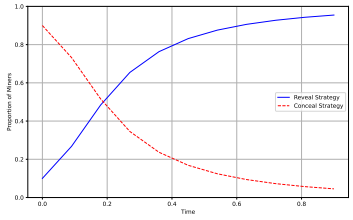
Here, we explain our results to highlight the strategic behavior of miners under different conditions. The analysis focuses on how the parameter $\phi(t)$, which serves as both a reward scaling factor and a vital component of the punishment system, influences the convergence of miners' strategies over time.

In analyzing the evolution of strategies among miners to counteract selfish mining, fixed points are identified where the strategy proportions stabilize $\frac{dx}{dt} = 0$. These fixed points represent stable equilibria in which miners' choices between concealing and revealing do not vary over time, signifying convergence to either strategy. The parameter $\phi(t)$ plays a significant role in influencing this convergence, modulating the reward-punishment mechanism and thereby shaping miners' strategic preferences across various scenarios. Figure 6.1 demonstrates this across values of $\phi(t)$ ranging from 0.1 to 0.9, with each scenario exhibiting distinct behaviors. Figure 6.2 indicates the evolution of the proportion of reveal and conceal strategies for different values of $\phi(t)$ over time.

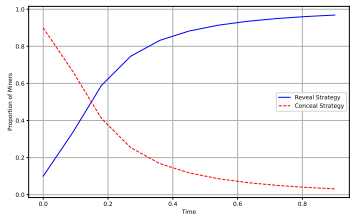
High Values of $0.9 \geq \phi(t) \geq 0.6$**:** As $\phi(t)$ continues to decrease, convergence occurs gradually, and the incentives for revealing increase. At $\phi(t) = 0.9$, only 35% of miners opt to reveal, showing a strong preference for concealment as severe punishment discourages transparency. As $\phi(t)$ decreases to 0.8, the proportion of revealing miners rises slightly to 42%, suggesting that even under elevated punishment, revealing can be favored under specific conditions. At $\phi(t) = 0.7$, the proportion increases further to 63%, where punishment levels still influence the decision, but miners are more inclined to reveal. By $\phi(t) = 0.6$, about 47% of miners reveal, indicating a near-equal preference for both strategies as transparency becomes more appealing. The decrease in the proportion of miners revealing from 63% at $\phi(t) = 0.7$ to 47% at $\phi(t) = 0.6$ may seem counterintuitive, as we might expect that lower $\phi(t)$ values would increase the incentive to reveal. However, this reduction can occur because, at this boundary, the incentives are balanced between revealing and concealing, creating an



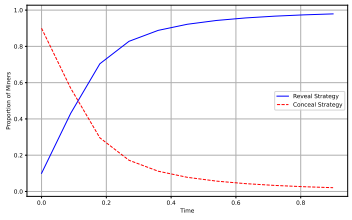
(a) Strategy evolution for $\phi(t) = 0.9$.



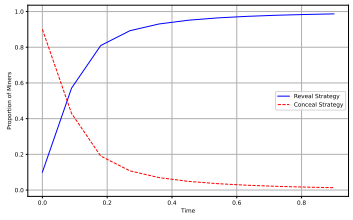
(b) Strategy evolution for $\phi(t) = 0.8$.



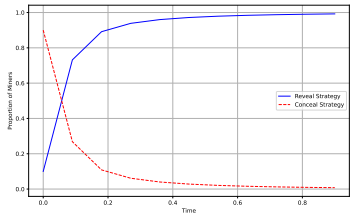
(c) Strategy evolution for $\phi(t) = 0.7$.



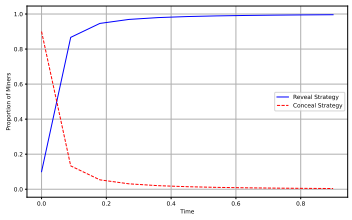
(d) Strategy evolution for $\phi(t) = 0.6$.



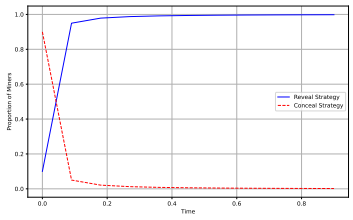
(e) Strategy evolution for $\phi(t) = 0.5$.



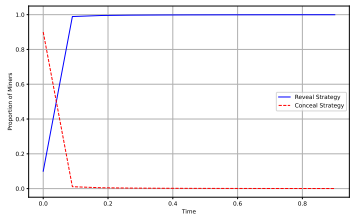
(f) Strategy evolution for $\phi(t) = 0.4$.



(g) Strategy evolution for $\phi(t) = 0.3$.



(h) Strategy evolution for $\phi(t) = 0.2$.



(i) Strategy evolution for $\phi(t) = 0.1$.

Figure 6.1 Strategy evolution for different values of $\phi(t)$.

equilibrium where concealment becomes more appealing even with decreased punishment.

Moderate values of $0.6 > \phi(t) > 0.3$: there is a noticeable shift towards increased adoption of the reveal strategy as $\phi(t)$ decreases. At $\phi(t) = 0.5$, around 61% of miners choose to reveal, indicating a preference for transparency despite moderate punishment. As $\phi(t)$ decreases to 0.4, the proportion of revealing miners rises to 76%, and by $\phi(t) = 0.3$, it reaches 88%. This trend suggests that lower values of $\phi(t)$ strengthen the incentives for honest behavior, revealing the dominant strategy.

Low values of $0.3 \geq \phi(t) \geq 0.1$**:** Convergence occurs almost immediately as miners quickly commit to their chosen strategies. At $\phi(t) = 0.3$, approximately 80% of miners favor the

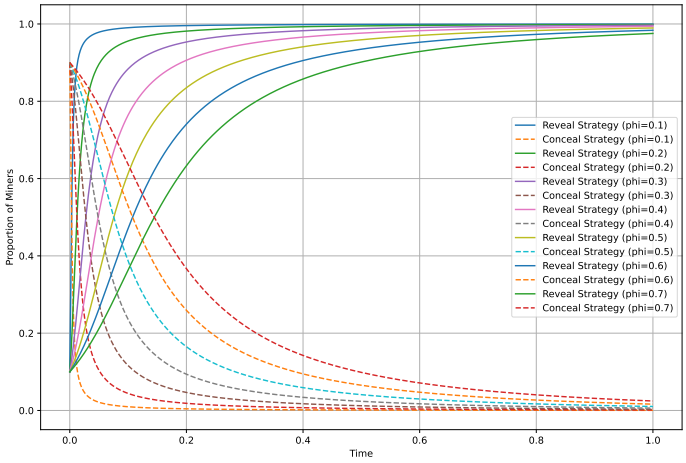


Figure 6.2 Evolution of proportion of reveal and conceal strategies for different $\phi(t)$ values over time.

reveal strategy, indicating a strong preference for transparency under moderate punishment conditions. As $\phi(t)$ decreases to 0.2, the proportion of revealing miners rises slightly to around 87%, and reaches approximately 95% by $\phi(t) = 0.1$. This trend suggests that lower values of $\phi(t)$ further encourage revealing, establishing transparency as the dominant strategy.

These findings underscore that $\phi(t)$ used in the reward and punishment mechanisms is critical in guiding miners' strategic preferences. Notably, moderate values of $0.6 > \phi(t) > 0.3$ emerge as a pivotal region where the game dynamics favor the *Reveal* strategy, with a significant number of miners opting for honest conduct. This suggests that these values of $\phi(t)$ effectively discourage selfish mining behaviors while promoting transparency. The existence of Evolutionarily Stable Strategies (ESS) within this range implies that the system achieves stable equilibria, where revealing is a more attractive strategy than concealing.

6.7 Conclusion

Selfish mining attacks represent a significant security threat to blockchain networks that allow miners to gain an advantage by holding and not broadcasting newly mined blocks, wasting honest miners' resources. First, this study determines the profitable time for the honest miner to stop the retaliation strategy. Then, we reduce the miners' indifference by utilizing a reputation system. Pursuing the mining process in pools with malicious miners should not continue to protect the resources of honest miners. Moreover, this research considered a reward-punishment system for miners who do not announce their block to the network

immediately after being mined will be punished, preventing them from adopting a subversive strategy. In other words, combining these two systems can prevent miners from executing a selfish mining attack as their revenue will decrease over time, and opting for a concealing strategy will not be profitable. In addition, this work leveraged a retaliation game, a war of attrition, to retaliate against selfish miners and discourage them from behaving selfishly. An evolutionary game approach is also employed to analyze how different reward and punishment structures influence the strategies taken by miners.

The results show that the reward scaling factor is crucial in directing the miners' tactics and balancing rewards and punishments effectively, which reduces incentives for selfish mining and enhances blockchain security. When more miners opt to reveal their blocks rather than keep them in their private chains, the game dynamics converge to stable equilibria where the reveal approach is favored. While promoting an honest strategy, the selected reward-punishment balance successfully reduces selfish mining behavior. In the future, we will explore adapting the retaliation game framework to PoS-based networks like Ethereum 2.0 to mitigate such threats.

CHAPTER 7 ARTICLE 4: SYBIL ATTACK DEFENSE IN BLOCKCHAIN-BASED INDUSTRIAL IOT SYSTEMS USING DECENTRALIZED FEDERATED LEARNING

Fatemeh Erfan, Martine Bellaiche, Talal Halabi

Submitted to Elsevier Journal of Internet of Things, June 6, 2025

Graphical Abstract

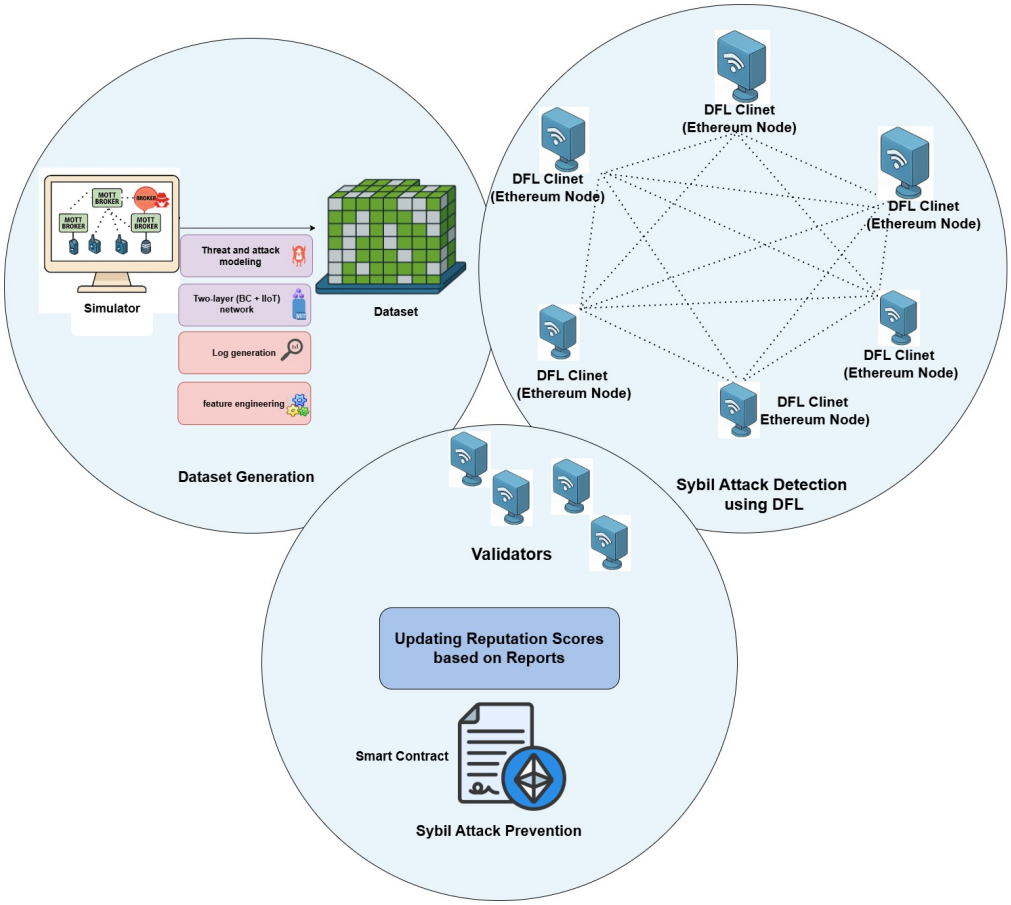


Figure 7.0 Graphical Abstract

Highlights

- Propose a privacy-preserving, decentralized federated learning framework for Sybil attack detection in blockchain-based IIoT networks.

- Introduce smart contract-driven reputation management for Sybil attack prevention.
- Develop and release a comprehensive Sybil attack simulator and labeled dataset for IIoT research.
- Demonstrate that DFL achieves high detection accuracy and superior resilience compared to centralized and traditional federated learning.

Abstract

Integrating blockchain into the Industrial Internet of Things (IIoT) has emerged as a promising solution for preserving data privacy and ensuring IoT security. Among various blockchain platforms, Ethereum stands out due to its support for smart contracts and its interoperability with lightweight communication protocols. Despite these advantages, particularly within Ethereum-based networks, IIoT systems remain vulnerable to large-scale threats such as Sybil attacks. These attacks pose a critical security risk because an adversary generates numerous fake entities to infiltrate and compromise the network, ultimately undermining its integrity and availability. Existing approaches utilize Ethereum smart contracts and lightweight protocols such as MQTT to secure IIoT communications, but often overlook sophisticated threats such as Sybil attacks, which introduce fraudulent nodes into the network. Conventional detection methods typically depend on centralized monitoring, undermining scalability and privacy, and there remains a lack of publicly available datasets representing adversarial behaviors in IIoT environments. In this paper, we first create an Ethereum-based IIoT network and release a publicly available dataset through our GitHub repository. Then, we propose an advanced method to detect and prevent Sybil attacks in a PoA-based IIoT network using decentralized federated learning. During the detection phase, we employ a convolutional neural network (CNN) within the decentralized federated learning framework, achieving an average detection accuracy and recall of 91.13% and 91.37% among clients, respectively. In the prevention phase, we design a secure smart contract that manages a dynamic reputation system, effectively preventing Sybil nodes from remaining active in the network.

Keywords: Sybil attack, Industrial Internet of Things, decentralized federated learning, threat detection, blockchain security, Ethereum networks

7.1 Introduction

The evolution of industrial paradigms and their integration with computer systems has profoundly transformed the current manufacturing landscape. The emergence of Industry 5.0 represents a significant shift from the purely technology-driven approach of Industry 4.0 to a

more human-centric and value-oriented framework. This next stage emphasizes the synergy between human creativity and advanced technologies, such as Artificial Intelligence (AI) and Internet of Things (IoT). In this context, IIoT plays a pivotal role by enabling intelligent, interconnected industrial systems capable of real-time data processing, automation, and autonomous decision making.

Unlike earlier generations that relied heavily on centralized architectures, IIoT under Industry 5.0 seeks to enhance resilience, sustainability, and customization in manufacturing by adopting decentralized and interoperable frameworks [194]. As industries increasingly integrate these advanced technologies, there is a growing demand to modernize workforce skill sets to ensure effective human-machine collaboration and uphold sustainable and socially responsible practices. However, traditional centralized IIoT architectures face significant limitations in terms of scalability, fault tolerance, and attack resilience—particularly under dynamic or adversarial conditions. Thus, decentralized and secure architectures, underpinned by blockchain and federated learning (FL), are becoming essential for enabling the resilience required in Industry 5.0 scenarios.

To meet these decentralization and security needs, blockchain (BC) technology has emerged as a foundational enabler for next-generation IIoT architectures [195]. IIoT generates vast amounts of data, and privacy and security are paramount. Thanks to the salient features of blockchain, such as decentralization, immutability, and transparency, the integrity and confidentiality of data transmitted across IIoT systems can be effectively preserved. Among various blockchain platforms, Ethereum has become one of the most widely adopted due to its superior capabilities, including fast transaction finality, support for smart contracts, and interoperability with external systems. The time required for a transaction to be confirmed and added to the Ethereum blockchain is significantly lower than that of Bitcoin, contributing to its popularity. Furthermore, Ethereum enables developers to implement custom smart contracts for various industrial tasks, including access control, automated payments, and secure data logging. In addition, Ethereum supports a broad range of interoperability protocols—such as Chainlink, InterPlanetary File System (IPFS), and Message Queuing Telemetry Transport (MQTT)—thereby enabling secure and trusted data exchange across devices, services, and blockchains. This interoperability is essential in IIoT applications where seamless communication across different factories, supply chains, and stakeholders is required.

Among consensus algorithms, Proof of Authority (PoA) is particularly well-suited for IIoT systems, outperforming alternatives like Proof of Stake (PoS) and Proof of Work (PoW) in resource-constrained environments. One of the key advantages of PoA is its minimal computational overhead, which allows lightweight IIoT devices and brokers to participate in

the network without the energy and processing demands associated with PoW-based systems. Moreover, PoA networks support horizontal scalability with minimal consensus delay, making them suitable for real-time industrial environments characterized by high-frequency data streams generated from sensors, programmable logic controllers (PLCs), and factory-edge devices [9].

As IIoT systems continue to evolve and scale, the integration of lightweight blockchain platforms such as Ethereum with efficient communication protocols like MQTT presents a compelling path toward building secure, scalable, and resilient industrial infrastructures. Despite the transparency, immutability, and resilience that blockchain offers IIoT infrastructures, several security challenges remain. A particularly critical threat is the Sybil attack, in which a malicious entity attempts to create numerous fake identities or nodes to gain disproportionate influence within the network. Once a Sybil attack is successfully executed, it can facilitate more severe attacks such as eclipse attacks, data spoofing, and manipulation of telemetry data, thereby compromising the reliability and safety of industrial operations.

To understand how these attacks propagate in practice, it is essential to examine the role of the MQTT protocol, the backbone of IIoT communication. MQTT is a lightweight, publish/subscribe messaging protocol designed explicitly for devices operating under constrained conditions such as limited bandwidth, low power availability, and intermittent network connectivity. These features make MQTT particularly suitable for IIoT scenarios, where numerous low-power devices, such as sensors and actuators deployed in smart factories, must continuously exchange telemetry data with centralized systems or decentralized edge brokers. Furthermore, the low overhead of MQTT makes it a practical choice for environments with restricted storage and computational capabilities, where traditional communication stacks may be infeasible [196].

IIoT systems typically operate across multiple architectural layers, each with distinct communication and security responsibilities. According to Leng et al. [13], IIoT systems can be abstracted into a four-layer architecture: the perception, interconnectivity, middleware, and data application layer. The *perception layer* includes physical sensors and devices that collect raw environmental or operational data. The *interconnectivity layer* ensures the reliable transmission of this data using lightweight communication protocols, such as MQTT, between sensors, gateways, and cloud or edge nodes. The *middleware layer* acts as a bridge that facilitates distributed computing, device interoperability, and secure execution environments. Finally, the *data application layer* handles data-driven services, such as analytics, monitoring, control logic, and machine learning inference.

Despite their modular design, current IIoT systems are constrained by the absence of robust,

secure, and scalable middleware platforms. Most middleware relies on centralized servers, which introduces Single Points of Failure (SPF), hinders horizontal scalability, and compromises the confidentiality and availability of sensitive industrial data [13, 197]. These limitations also reduce the system's adaptability in the face of sophisticated threats, such as Sybil attacks and data spoofing. As a result, researchers have increasingly advocated for decentralized middleware architectures that integrate blockchain for tamper-resistant communication and FL for privacy-preserving intelligence across distributed IIoT nodes. Among the critical security threats that challenge the integrity and resilience of decentralized IIoT systems, Sybil attacks remain particularly concerning due to their ability to exploit identity and trust mechanisms.

There are various types of defenses proposed against security attacks in blockchain-based IoT systems [67, 198]. This work focuses on the underexplored challenge of Sybil attack detection in MQTT-based IIoT networks. Here, a Sybil attack refers to a security breach in which a malicious entity generates multiple counterfeit MQTT brokers, commonly called Sybil nodes. The primary objective of such an attack is to disrupt the integrity and reliability of the network by impersonating legitimate brokers. These fake nodes can transmit falsified sensor data to interfere with normal operational processes. They also flood the network with deceptive messages, leading to communication overload and exerting undue influence on the message propagation mechanisms within the IIoT system. Such attacks may be perpetrated by external adversaries or insiders with authorized access, posing significant risks to the security, performance, and trustworthiness of the IIoT infrastructure.

While MQTT is widely adopted for lightweight messaging in IIoT environments, it lacks strong built-in security features [199]. To address these limitations, blockchain technology offers a promising alternative by enabling decentralized, transparent, and tamper-resistant data management. Despite these advantages, the Sybil attack remains a persistent challenge, even in blockchain-enabled IIoT systems. Beyond architectural and protocol-level challenges, researchers face a significant bottleneck in the form of limited public datasets tailored to Sybil attack detection in IIoT environments. Most existing datasets either lack contextual relevance to industrial settings or fail to capture the diverse and dynamic communication patterns exhibited in IIoT networks that rely on protocols like MQTT.

Detecting Sybil attacks within IIoT environments presents several critical challenges. First, the dynamic communication patterns of MQTT brokers complicate detection efforts. These brokers exhibit highly transient behavior, frequently establishing and terminating connections, which makes it difficult to distinguish between normal activity and suspicious anomalies. Second, data privacy constraints inherent in industrial systems limit the extent to which

operational data can be shared or centralized for analysis. This restriction poses significant obstacles to collaborative or federated detection mechanisms. Third, adversarial mimicry presents a sophisticated threat, as attackers may deliberately emulate the communication patterns and behaviors of legitimate MQTT brokers. This strategic imitation allows malicious nodes to evade traditional detection methods that rely on behavioral profiling or signature-based analysis.

Our proposed defense system uses a Sybil attack detection model to identify malicious MQTT brokers acting as Ethereum nodes. The model focuses on detecting peer discovery anomalies, duplicate identities, and suspicious connection patterns that indicate the presence of Sybil nodes. This approach leverages Decentralized Federated Learning (DFL) to ensure scalable and privacy-preserving detection across the distributed IIoT network. The key contributions of this research are as follows.

- **Privacy-Preserving Decentralized Detection via Federated Learning:** We propose a secure and privacy-preserving DFL framework for Sybil attack detection in blockchain-enabled IIoT networks. In this architecture, each MQTT broker retains its raw data locally and participates in collaborative detection by sharing only encrypted or summarized model updates. This decentralized, hierarchical learning approach not only preserves data confidentiality and adheres to industrial privacy constraints but also enhances scalability and detection accuracy across distributed brokers.

- **Smart Contract-Based Reputation Management:** We design a smart contract to manage reputation scores of MQTT brokers, computed based on peer discovery activity, message patterns (such as unusual `PUBLISH`/`SUBSCRIBE` behavior), and connection frequency. Brokers with a low reputation are flagged as potentially malicious and can be restricted via a decentralized application (DApp) interface.

- **Sybil Attack Simulation and Evaluation Framework:** We develop a comprehensive Sybil attack simulator tailored for PoA-based Ethereum IIoT networks, enabling realistic emulation of multi-protocol communication (e.g., MQTT, devp2p, discovery, libp2p, ENR). This simulator facilitates the generation of a rich and labeled dataset by extracting critical behavioral features from communication logs. Using this dataset, we train a CNN-based detection model under three settings: centralized deep learning (DL), FL, and DFL. Through empirical evaluation, we demonstrate that while all models achieve high detection accuracy, DFL offers distinct advantages. It preserves data privacy by retaining data locally, eliminates the need for a central aggregator, and provides enhanced robustness against adversarial manipulation. These architectural

and security benefits make DFL particularly suitable for trust-sensitive, decentralized IIoT environments.

The rest of the paper is structured as follows. Background concepts are explained in section 7.2. Section 7.3 presents the existing works in this field. Our proposed approach is presented in Section 7.4. In Section 7.5, the solution is evaluated and the results are discussed. Finally, Section 7.6 concludes the paper.

7.2 Background Concepts

This section provides background information on blockchain fundamentals, communication protocols used in blockchain-based IIoT networks, and the federated learning technique.

7.2.1 Blockchain Concepts

Blockchain is a decentralized system that allows transactions to be verified by a group of participants [198]. This part provides an overview of the fundamental concepts underlying blockchain technology.

Block. In blockchain systems, transactions are grouped into units called blocks, which are sequentially linked to form a distributed ledger, starting with the genesis block. Each block typically contains a header and a body: the body holds the transaction data, organized in a Merkle tree structure that ensures data integrity through hash-based aggregation. The Merkle root, computed from individual transaction hashes, enables efficient verification of block content [200]. The block header includes metadata such as the block's identifier, timestamp, reference to the previous block, and the Merkle root. In a PoA blockchain, block generation does not rely on solving cryptographic puzzles but is instead managed by pre-approved validator nodes. These validators are responsible for appending new blocks according to a deterministic or scheduled order, ensuring both transparency and efficiency while maintaining trust through identity verification mechanisms [29, 201].

Validator. In PoA-based blockchains, the role traditionally referred to as a miner in PoW systems is fulfilled by a validator. Validators are known, trusted entities authorized to create new blocks and validate transactions. Unlike PoW and PoS systems, where participants exploit their hashing power or stake tokens, validators in PoA networks are selected based on their identity and reputation. Each validator is assigned the task of proposing and sealing new blocks in a predefined or rotating sequence. This approach eliminates the need for resource-intensive operations, significantly reducing energy consumption and latency, factors critical

to the performance of Industrial IoT environments [202, 203]. Validators must maintain a high level of availability and honesty, as any detected misbehavior can lead to revocation of their authority within the network [204].

Consensus Algorithm. The consensus algorithm ensures that all participants in a blockchain network agree on the current state of the ledger. In PoW, participants solve cryptographic problems using significant computational resources to validate and append new blocks. In contrast, Ethereum transitioned to PoS in 2022 to reduce energy consumption and improve scalability [188]. While both PoW and PoS remain susceptible to selfish mining attacks [198], their mechanisms differ fundamentally from those employed in permissioned blockchain settings. PoA, on the other hand, is designed specifically for private or consortium networks. In PoA, a small group of pre-selected and trusted validators is authorized to propose and validate new blocks. Consensus is reached through a deterministic or rotating process among these validators, eliminating the need for competition or staking. This leads to significant improvements in throughput, latency, and energy efficiency—making PoA especially suitable for IIoT applications. However, PoA also introduces governance challenges, such as managing validator trust and ensuring fault tolerance in partially synchronous networks [201, 205].

7.2.2 Communication Protocols

In blockchain-based network architectures, the peers are connected through several communication protocols that serve different layers of the system. These are described below.

Gossip-like Propagation Protocol

In Ethereum-based PoA networks, a gossip-like mechanism is employed to propagate blocks and transactions across the peer-to-peer (P2P) network. Although it does not use `libp2p`'s GossipSub, the underlying `devp2p` protocol provides similar functionality to maintain consistency and synchronization. Messages exchanged at this layer include `NewBlock`, which distributes newly mined blocks to peers; `NewBlockHashes`, which announces the existence of new blocks without transmitting the full block data; and `Transactions`, which broadcasts new transactions to the network. An attacker may exploit these messages by broadcasting malicious or fake blocks and transactions from multiple Sybil identities to overwhelm the network and disrupt consensus.

DevP2P Protocol

The Ethereum devp2p (Developer P2P) protocol forms the core communication framework in Ethereum 1.x clients such as Geth. It is responsible for peer discovery, connection management, message propagation, and data synchronization. Relevant message types monitored for anomaly and Sybil detection include:

- Peer Discovery: Includes mechanisms for identifying and maintaining connections with new peers. Ethereum uses the Node Discovery Protocol v4 (based on Kademlia), which exchanges messages such as `Ping`, `Pong`, `FindNode`, and `Neighbors`.
- Blockchain Synchronization: This includes the exchange of headers, bodies, receipts, and state data using messages such as:
  - `GetBlockHeaders`, `BlockHeaders`: Request and return block headers.
  - `GetBlockBodies`, `BlockBodies`: Request and return full block contents.
  - `GetReceipts`, `Receipts`: Used to retrieve transaction receipts.
  - `GetNodeData`, `NodeData`: Request and transmit Ethereum state data.
  - `GetProofs`, `Proofs`: Request and return Merkle proofs for efficient verification.
- Transactions and Blocks: These messages are used to propagate transactions and new blocks across the network including `Transactions`, `NewBlock`, and `NewBlockHashes`.
- Connection Management and Handshakes: These messages facilitate session initialization, capability negotiation, and peer management to maintain stable connections.

These messages form the basis of P2P Ethereum networking in PoA and are essential for maintaining node synchronization and state consistency. Monitoring them is crucial for detecting unusual propagation patterns or coordinated Sybil behaviors.

Node Discovery Protocol

The Ethereum Node Discovery Protocol (v4) is used to help nodes locate other peers in the network. It uses a UDP-based Kademlia-like distributed hash table (DHT) for efficient lookups. Message types include: `Ping`, `Pong`, `FindNode`, and `Neighbors`. A Sybil attacker can exploit this protocol by registering multiple fake nodes, misleading honest peers about the network topology and inflating their influence.

Ethereum Node Records and Discovery v5

In addition to Discovery v4, Ethereum also supports Discovery v5 and Ethereum Node Records (ENRs) [206]. Discovery v5 introduces encrypted UDP packets and advanced message types including `NODES`, `TALKREQ`, `TALKRESP`, and `TOPICQUERY` [207]. ENRs allow nodes to share metadata in a signed, self-contained record. Sybil nodes may exploit ENRs by generating multiple records with similar or spoofed properties, such as IP address or sequence number.

MQTT Protocol

One of the most prominent protocols utilized in IIoT systems is MQTT thanks to its lightweight design, low bandwidth requirements, and efficient communication model. In blockchain-based IIoT, MQTT is used to enable communication between sensors, PLCs, and the Ethereum node-based brokers. The main MQTT message types include:

- `CONNECT`, `CONNACK`: Used for establishing a session between the client and broker.
- `PUBLISH`: Sends messages from a client to the broker.
- `SUBSCRIBE`, `SUBACK`: Used for subscribing to topics and acknowledgment.
- `UNSUBSCRIBE`, `UNSUBACK`: Used for removing topic subscriptions.
- `PINGREQ`, `PINGRESP`: Provides the keep-alive functionality.
- `DISCONNECT`: Used for graceful session termination.

libp2p Networking Library

`libp2p` is increasingly adopted in Ethereum 2.0 and other decentralized systems as a modular P2P networking stack [208]. It provides advanced message propagation mechanisms through control messages such as `GRAFT`, `PRUNE`, `IHAVE`, and `IWANT`, which are essential for maintaining dynamic gossip mesh overlays. While these control flows are not currently integrated into PoA-based Ethereum clients like Geth, their growing relevance in modern decentralized architectures highlights the importance of future compatibility and monitoring capabilities.

7.2.3 Federated Learning

Machine learning (ML) can be categorized into three primary types. The first type is localized learning, where the model is trained directly at an individual endpoint using its own

data. Centralized learning is another type of machine learning, in which data from various endpoints is gathered and used to train the model at a central location. The last type is FL, where the model is collaboratively trained across multiple endpoints, utilizing their local data sets without requiring direct data exchange. Each of these approaches has its own set of advantages and challenges [95].

FL offers a decentralized method for building machine learning models by keeping data on local devices rather than transferring it to a central server. In this approach, participating endpoints train models locally, and only the updated model parameters are shared with a central entity, which consolidates these updates to form a global model. This technique significantly enhances privacy, as raw data never leaves local devices. Moreover, computation, communication, and storage overheads are decreased owing to the decentralized learning architecture, which obviates the need for transferring or aggregating training datasets at a central server [95].

Despite the notable advantages of FL, it remains susceptible to cyber threats at the central server, which coordinates model aggregation. To address this limitation, DFL has been introduced. In DFL, each participating node independently trains a local model on its private data and engages in P2P communication with other nodes to exchange and aggregate model updates along with relevant metadata (e.g., neural network activation functions).This decentralized exchange mechanism addresses several limitations inherent in centralized FL, including the risks associated with SPF, reliance on a central coordinating entity, and communication bottlenecks at the server level [96].

7.3 Related Work

This section presents the existing works in Sybil attack defense and provides a comparative analysis as shown in Table 7.1.

Alrubei et al. [154] present a hybrid consensus algorithm that integrates lightweight PoW with PoA by using honesty scores derived from device activity to select trustworthy validators in IoT networks. This approach aims to evaluate and compare the effectiveness of PoW and PoA consensus mechanisms in preventing Sybil attacks within wireless sensor networks by simulating attack scenarios and measuring blockchain performance and security under each approach. However, the mechanism imposes additional computational overhead on constrained devices due to the need for continuous score generation and secure scoring infrastructure, potentially affecting energy efficiency and scalability.

Kokate et al. [155] introduce a trust-weighted PoA consensus for secure healthcare IoT

Table 7.1 Comparison of most recent Sybil attack mitigation techniques in blockchain-based IIoT/IoT.

Reference	Domain	Consensus / Mechanism	Technique	Defense Type	Limitations
Mishra et al. (2018) [161]	IoT	Markov Chain + K-Means	Analytical model of Sybil behavior and stealth strategy	Prevention	Not implemented; unrealistic attacker assumptions
Alrubei et al. (2022) [154]	IoT	HDPoA (PoW + PoA)	Lightweight PoW for honesty score, used in PoA selection	Prevention	Additional computation and complexity in score management
Eisenbarth et al. (2022) [160]	Ethereum	Centralized Crawler + Smart Contract	Periodic detection via crawler, reporting via smart contract	Detection	Centralized monitoring; gas cost; limited real-time response
Sabrina et al. (2022) [156]	IoT	Permissioned BC	Device identity management using fuzzy extractors	Prevention	Requires pre-established identity infrastructure
Wang & Tan (2023) [157]	IoT / BC	PBFT with Reputation	Reputation scoring to weight PBFT votes	Detection	SPF; communication overhead
Dewanta et al. (2023) [159]	IoT	Lightweight Crypto + Timestamps	Symmetric encryption + timestamp verification for message integrity	Prevention	SPF at broker; weak key distribution assumptions
Venkatesan et al. (2024) [158]	IIoT	Hybrid + ML	Hybrid consensus with ML-based Sybil detection	Detection	complex; high latency; SPF in some components
Prathiba et al. (2024) [163]	IIoT	NFTs + DTs + Smart Contract	Unique NFT identities + DTs validation for model integrity	Prevention	High computation/storage costs; static DTs and centralized init
Zhao et al. (2025) [162]	IIoT / Retail	Reputation System + Tokenization	Purchase-tax-linked tokens for reputation scores	Prevention	Relies on tax authority; adds latency and scalability limits
Kokate etal. (2025) [155]	IoT / HCIoT	Trust-weighted PoA	Trust score modifies validator power	Prevention	Trust bias risk; SPF

(HCIoT) systems, where validator selection is influenced by dynamic trust metrics to prevent Sybil nodes from gaining influence. Nonetheless, the approach relies on accurate trust assessment and central management of trust scores, making it vulnerable to the risks of bias, manipulation, and SPF in validator ranking.

Sabrina et al. [156] design an IoT data-sharing framework based on Ethereum smart contracts, that prevents Sybil attacks by leveraging fuzzy extractors for device identity binding to ensure only authenticated nodes participate in the blockchain. Despite its strong identity assurance, the system assumes the existence of a robust and scalable identity infrastructure, creating complexity in key management and potential barriers to deployment in large-scale, heterogeneous IoT ecosystems.

Wang and Tan [157] illustrate a reputation-enhanced Practical Byzantine Fault Tolerance (PBFT) consensus mechanism that assigns greater influence to nodes with verified behavioral integrity, thereby suppressing Sybil identities in voting rounds. The proposed model relies on a single master node selected based on the highest reputation, which introduces an SPF in the consensus process. If this primary node is compromised or fails, it can disrupt consensus or be exploited to manipulate the network. It also increases communication overhead due to message tracking and auditing.

Venkatesan and Rahayu [158] propose a blockchain framework by combining hybrid consensus algorithms, including PoW, PoS, DPoS, PBFT, and PoCASBFT, with machine learning techniques to detect and prevent various cyberattacks, such as Sybil and 51% attacks. This

approach faces higher latency due to PBFT and DPoS components, which have known latency and message complexity issues, especially as the network size increases. While innovative and promising, its complexity, potential overhead, and centralization tendencies in some consensus combinations present real-world deployment challenges, particularly in IIoT environments where resources and reliability are critical.

Dewanta et al. [159] propose a lightweight cryptographic protocol to protect MQTT-based IoT communications against Sybil attacks. The scheme enhances message security through timestamp-based freshness verification, session key derivation using hash functions, and symmetric encryption of message payloads. It aims to ensure confidentiality, integrity, and replay resistance while maintaining low computational overhead, demonstrating feasibility on resource-constrained IoT devices. However, the approach relies heavily on the integrity of the MQTT broker and the secure management of private keys, introducing an SPF if the broker is compromised or keys are exposed. Furthermore, it assumes the secure pre-distribution of cryptographic parameters, such as private keys and hash algorithms, without offering a scalable or robust key exchange mechanism, limiting its applicability in dynamic or large-scale IoT environments.

Eisenbarth et al. [160] propose a Sybil prevention framework based on a centralized monitoring system that employs a custom crawler to identify suspicious nodes and disseminates this information via smart contracts on the Ethereum blockchain. While effective in detecting large-scale identity forgeries, the system introduces an SPF, incurs substantial on-chain gas costs for publishing NodeID data, and lacks native client-side enforcement, relying instead on external tools for manual revocation. Moreover, its reliance on periodic crawls limits real-time responsiveness. In contrast, our DFL approach enables scalable, privacy-preserving, and automated Sybil detection in PoA-based IIoT environments, allowing each node to independently assess peer behavior and contribute signed severity reports to an on-chain smart contract for real-time reputation updates and trust enforcement.

Mishra et al. [161] propose an approach for mitigating Sybil attacks in IoT networks by capturing the behavior of the attackers through a Markov chain during the compromise phase, employing K-means clustering to determine strategic deployment locations, and utilizing an identity replacement policy to sustain stealth during the launch phase. Despite its analytical depth, the proposed framework remains largely theoretical and has not been evaluated within practical IoT deployments, raising concerns about its real-world effectiveness. Moreover, the model presumes extensive knowledge and capabilities on the part of the attacker—such as node energy states and geographic proximity, which may not be readily available in real attack conditions, thereby limiting its generalizability.

Zhao et al. [162] propose a blockchain-based reputation framework for IIoT retail environments, utilizing taxed-purchase tokens and cryptographic authentication to ensure that only verified entities can contribute to reputation scores, thereby reducing the risk of Sybil attacks. The system fundamentally relies on the existence of a reliable and incorruptible tax authority capable of issuing and verifying taxed transactions. In the absence of such institutional integrity, the system's trust model may be compromised, rendering it ineffective. Furthermore, the incorporation of blockchain operations introduces performance and scalability limitations; as transaction volumes grow, processing delays and decreased responsiveness may negatively impact user experience and hinder widespread adoption.

Prathiba et al. [163] design a Secure Federated Learning (SFL) framework for IIoT environments by integrating Non-Fungible Tokens (NFTs) and Digital Twins (DTs) to enhance authentication and model integrity. NFTs, generated using EdDSA signatures, ensure unique participant identities to prevent Sybil attacks, while DTs serve as static benchmarks to detect and recover from data or model poisoning during federated learning aggregation. However, this framework introduces significant computational and storage overhead due to the combination of blockchain logging, NFT generation, smart contract-based model validation, and digital twin benchmarking, which may not scale efficiently in resource-constrained IIoT deployments. Furthermore, the approach relies on a centralized initialization phase and a static digital twin, making it vulnerable to an SPF. Lastly, frequent blockchain transactions and EdDSA-based identity checks, although secure, incur non-negligible gas and processing costs, potentially affecting throughput and real-time responsiveness in large-scale IIoT networks.

7.4 The Proposed Approach

This work presents a decentralized framework for detecting and preventing Sybil attacks in IIoT networks using decentralized federated learning combined with Ethereum smart contracts. The defense system leverages hostoric network information, Sybil indicators, flags, and signs of malicious activities, and then use them to prevent Sybil nodes from launching a potential attack. The system leverages key communication protocols and anomalous behavioral patterns for local anomaly detection. Model updates are trained and shared among MQTT-integrated Ethereum nodes in a privacy-preserving and tamper-proof manner. Specifically, first, we run a one-time Sybil attack on the network and train our model to detect Sybil nodes with high accuracy. Then, we use this model to observe the network and give a severity score to each node and report to the reputation management system written on a smart contract to prevent the launch of Sybil attacks. Therefore, this approach can be used as a separate detection unit in the network to detect Sybil nodes, as well as a prevention

unit to maintain the security of the network against Sybil attacks. In the detection unit, decentralized federated learning is applied to detect Sybil nodes. The results will be used in the prevention phase, where the reputation management system works with the results of the detection phase and scores the nodes based on those results. Each suspicious node is penalized and its score is dropped.

In this section, first, we illustrate our system architecture. Then, the threat model is described. After that, the simulation phase is explained, and then the detection and prevention units are presented.

7.4.1 System architecture

The reputation management and detection system operates in a heterogeneous IIoT environment containing: MQTT brokers with an Ethereum address and running local machine learning training as part of federated learning; Resource-constrained IoT devices; an MQTT messaging layer for lightweight device communication (with MQTT brokers managing publish/subscribe topics); a permissioned Ethereum PoA blockchain running by a consortium of IIoT gateways or authority nodes, hosting smart contracts for identity management and reputation tracking; and a Federated Learning coordinator, which is decentralized across the PoA validators or implemented as a service that interacts with the blockchain.

Reputation tracking serves as the critical link between detection and prevention: it transforms the process from merely identifying indicators of Sybil nodes to actively enforcing defenses in a decentralized and scalable manner. Without an integrated reputation mechanism, even the most sophisticated detection methods would be insufficient, as attackers could continue to operate within the network after being flagged.

Figure 7.1 illustrates the high-level architecture, which includes Ethereum nodes, MQTT brokers, smart contracts, Sybil attackers, PLCs, and sensors.

MQTT brokers play a critical role in the detection and prevention phases. First, they act as an MQTT message broker for nearby IoT devices. In the detection phase, MQTT brokers monitor all types of protocol messages—MQTT, ENR, discovery, libp2p, and devp2p, to extract behavior-based features and detect abnormal or Sybil behaviors. In addition, they run local machine learning training using locally received data and participate in decentralized FL by sharing and evaluating models with peer brokers. In the prevention phase, they hold an Ethereum identity (public/private key) to interact with smart contracts on a private PoA Ethereum blockchain and manage a reputation system based on the results achieved in the previous phase.

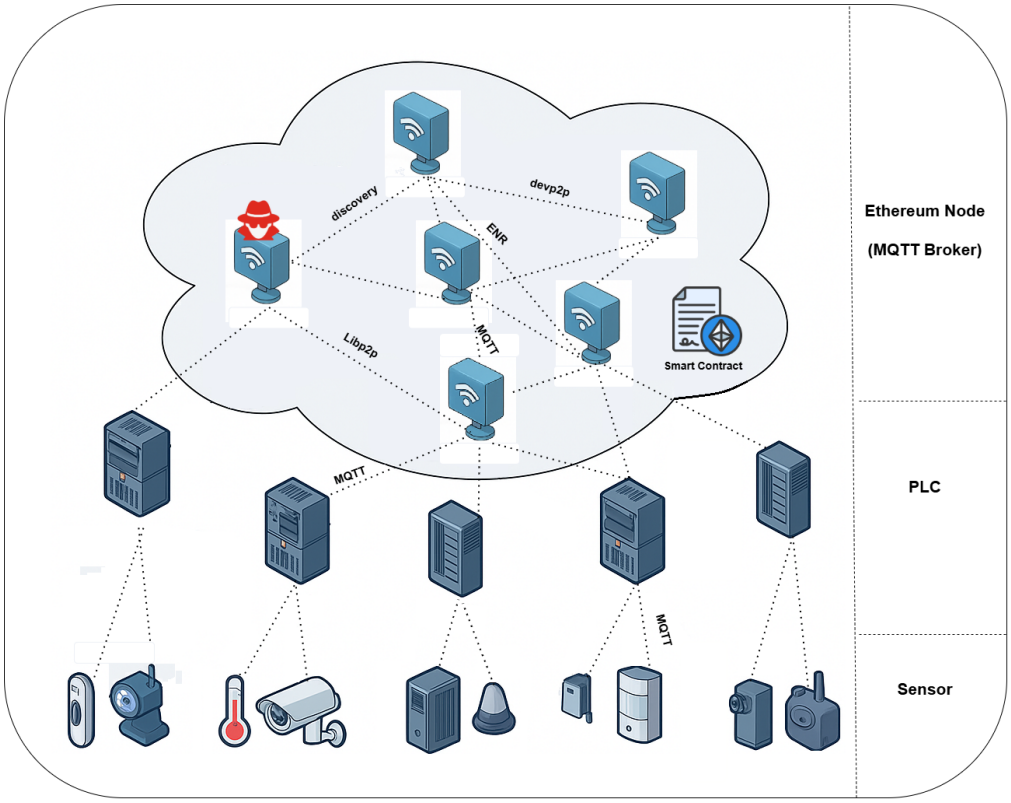


Figure 7.1 Ethereum-based IIoT architecture.

IoT devices including (PLCs, sensors) are lightweight, resource-constrained devices that publish MQTT messages (e.g., sensor readings, event logs) to their nearest MQTT broker. They do not participate in model training but serve as the data source for learning tasks.

The communication layer combines both MQTT and Ethereum networking protocols. The MQTT protocol facilitates lightweight communication between IoT devices and MQTT brokers using publish-subscribe channels. It also supports broker-to-broker messaging for decentralized coordination. The Ethereum networking protocols consist of *ENR, libp2p, devp2p*, and *discovery layers*. This hybrid protocol environment enables multi-modal message monitoring, allowing the system to analyze behavioral patterns not only in MQTT but also in Ethereum peer discovery, block propagation, and routing updates—offering deeper visibility into Sybil tactics. All communications between entities are explained in Table 7.2.

The PoA-based Ethereum blockchain hosts smart contracts which maintain reputation scores for each broker node and enforce access control policies including blacklisting low-reputation nodes, and prioritizing updates from high-reputation peers.

Table 7.2 Communication and protocol mapping in an IIoT architecture.

Communication	Protocol	Description
Sensor to Sensor	MQTT	Direct communication for sharing information.
Sensor to PLC	MQTT	Sensors publish data to the brokers for PLC access.
PLC to Sensor	MQTT	PLCs send commands to the sensors.
PLC to Brokers	MQTT	PLCs publish data/commands to the brokers.
Brokers to PLC	MQTT	Brokers send messages to the PLCs based on subscriptions.
Brokers to Brokers	devp2p, ENR, MQTT, Libp2p, discovery	Brokers share state and data using multi-protocol communication.

7.4.2 Threat Model

Here, we describe the attack in detail as well as the adversary's objectives and assumptions. We also discuss the rationale for adopting a decentralized defense approach. Finally, we explain the observable indicators characterizing Sybil behavior.

Attack description

Maintaining the stability and performance of IoT systems against large-scale security attacks is one of the critical challenges today, since the system components are highly interconnected, meaning a failure in one can cascade and impact others. One of the dangerous types of security attacks is Sybil attacks due to the ease of launch and its high potential impact. A Sybil attack [56] involves an adversary creating multiple fake identities to gain disproportionate influence or disrupt the network. Figure 7.2 illustrates how an attacker compromises the network through a Sybil attack. These attacks manifest as:

- Multiple Fake Clients: An attacker could create many fake MQTT clients that connect to the broker, potentially overwhelming the broker or misleading other clients.
- Fake Ethereum Nodes: In the Ethereum context, a Sybil attack may involve an attacker creating multiple fake nodes to try to influence the blockchain network or mislead the system.

TLS/SSL encryption in MQTT settings is commonly used to safeguard message exchanges between clients and brokers by ensuring data confidentiality during transmission. However, although this encryption method can defend against interception, unauthorized modifications, and intermediary interference, it remains ineffective in addressing identity-based threats such

as Sybil attacks [159]. Given the limitations of traditional cryptographic techniques and centralized defense mechanisms, there is an increasing need for scalable, privacy-preserving, and distributed detection strategies. Federated learning, particularly in its decentralized form, has emerged as a promising solution for collaboratively identifying Sybil behaviors across IIoT networks without compromising data privacy.

Adversarial goals and assumptions

In the proposed architecture, we assume the presence of an adversary or a coordinated group of adversaries attempting to launch a Sybil attack by injecting multiple fake identities into a PoA-based IIoT network. These Sybil nodes may originate from external sources or result from compromised internal entities. The objective of the attacker is to disrupt normal system operations and degrade the integrity of the DFL process by submitting manipulated or malicious model updates. Ultimately, such actions aim to corrupt trust within the network, degrade the reputation of honest brokers, facilitate the injection of malicious transactions, and compromise the overall security of the system. To evade detection, the attacker can generate anomalous traffic patterns that closely mimic legitimate behavior.

In addition, we assume that the majority of brokers within the network operate honestly and that the underlying blockchain infrastructure is securely configured. Reputation scores are initialized in a fair and unbiased manner, and brokers are sensitive to privacy and abstain from sharing raw data. Moreover, within the DFL framework, only pre-authorized nodes that are nominated as validators are allowed to submit reports to the smart contract-based reputation management system.

Under these conditions, our approach provides effective protection against both Sybil attacks and model poisoning attempts perpetrated by Sybil nodes, thereby ensuring robust detection mechanisms and reliable reputation management. To detect such Sybil behaviors in practice, we rely on observable communication patterns and protocol-level indicators that distinguish malicious nodes from honest participants.

Observable indicators of the Sybil behavior

A Sybil attacker may also create numerous fake MQTT clients with distinct client IDs that subscribe or publish falsified sensor data in rapid succession, overwhelming the broker and affecting decision-making processes. Prior research has shown that protocol message monitoring, combined with behavioral analysis, enhances the ability to detect Sybil attacks in P2P systems. In our defense solution, we propose to monitor the relevant indicators, including:

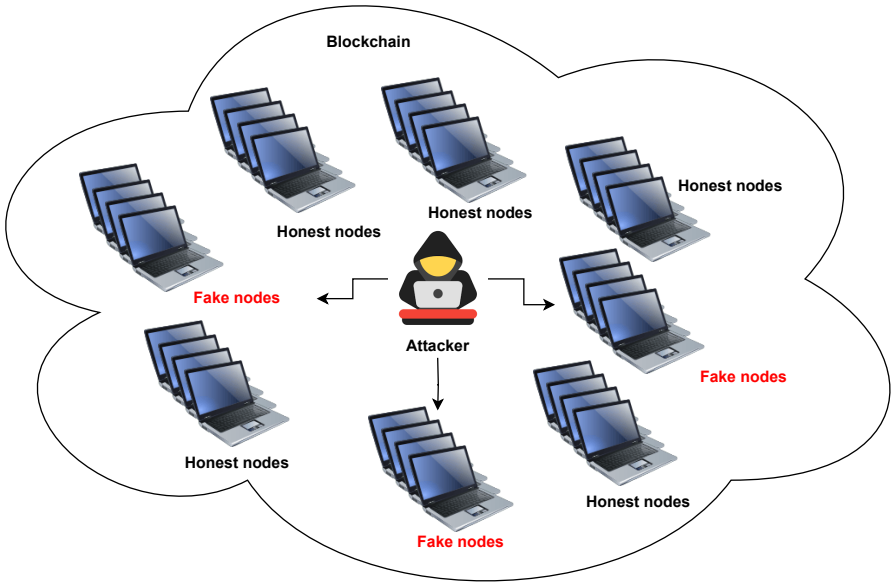


Figure 7.2 Sybil threat model.

- Dominant protocol ratio: Sybils often overuse a few protocols (e.g., spammers overusing MQTT:PUBLISH) [209].
- Protocol entropy: Low entropy may signal repeated message types (i.e., non-diverse behavior) [210].
- Latency spike ratio: Unusually high latency may indicate resource exhaustion or deceptive delay tactics [211].
- Peer overlap ratio: Measures how many distinct peers a node communicates with in a window. Useful for detecting clustered Sybil behavior [212].
- Message Frequency and Bursts: Sybil nodes may produce bursts of `PUBLISH`, `PINGREQ`, or `GetBlockHeaders` messages [209].
- Timing anomalies: Identical message timing from different IDs suggests centralized control.
- Topology manipulation: Returning highly redundant `Neighbors` or ENRs [209].

Tracking such statistical behaviors provides strong evidence for distinguishing Sybil nodes

from legitimate participants. In Section 7.4.3, we operationalize these patterns using 16 engineered features extracted from multi-protocol communication logs.

7.4.3 Dataset Generation and Processing

According to [213], a robust simulator must support large-scale network models with realistic traffic patterns to accurately reflect real-world scenarios. It should also be capable of launching scalable attacks and supporting multiple communication protocols to closely approximate the complexity of real networks. Our simulator fulfills these criteria by enabling the configurable deployment of both honest and Sybil nodes within a PoA-Ethereum-based IIoT environment. It supports a variety of communication protocols, including MQTT, devp2p, discovery v4, libp2p, and ENR exchanges, thereby capturing the multi-protocol nature of industrial systems. Furthermore, it facilitates the simulation of coordinated Sybil attacks through programmable behaviors such as spamming, flooding, and evasion, which can be dynamically triggered and scaled across multiple Sybil clones. The simulator logs per-node message traffic in a time-ordered format, enabling detailed behavioral analysis and temporal pattern recognition.

Beyond protocol-level fidelity, the simulator extracts 16 engineered features for each node over a sliding communication window. These features include, but are not limited to, protocol entropy, dominant protocol ratio, message burstiness, latency patterns, and peer interaction metrics. The data is exported in both raw JSON and structured CSV formats, with ground truth labels identifying each node as either Sybil or honest. This feature-rich, labeled dataset facilitates rigorous training and evaluation of machine learning models for Sybil attack detection. Integrating multi-layer protocol modeling and realistic adversarial behaviors, the simulator provides a solid foundation for studying Sybil resilience in blockchain-enabled IIoT networks.

The workflow of the simulator, illustrated in Figure 7.3, is categorized into three primary phases: data generation, data collection and feature extraction, and dataset preparation.

Data generation:

This step consists of two main components, threat and attack modeling and network configuration, which are explained below.

Threat and attack modeling: The simulation environment includes a configurable number of honest and Sybil nodes, each uniquely identified and behavior-labeled. Honest nodes operate with protocol-compliant communication, while Sybil nodes are assigned one of three

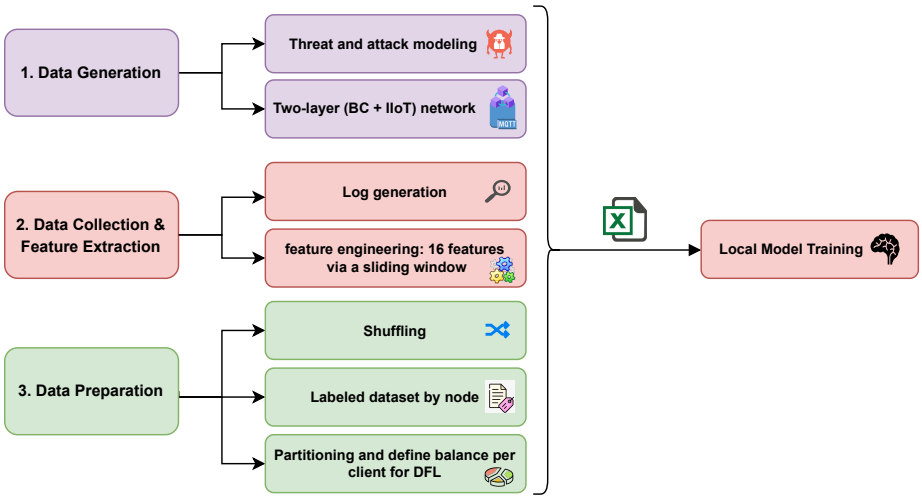


Figure 7.3 The dataset generation and processing phase.

predefined malicious strategies:

- Spammer: Sybil nodes repeatedly transmit high-frequency `MQTT:PUBLISH` messages to overload brokers with falsified sensor data, imitating flooding attacks observed in MQTT-based IIoT systems [209].
- Flooder: Malicious nodes broadcast redundant or unsolicited protocol messages such as `NewBlock`, `Ping`, or `FindNode`, exploiting Ethereum's peer discovery and block propagation channels to overwhelm honest participants [211].
- Evasive: Nodes manipulate Ethereum Node Records (ENRs) or discovery responses to evade detection by appearing similar to legitimate nodes. Such tactics have been documented in Sybil evasion models leveraging spoofed ENR fields and timing camouflage [209, 210, 214].

Network configuration: The simulator models a two-layer network architecture:

- The *blockchain layer* includes peer discovery and propagation protocols such as `libp2p`, `devp2p`, `discovery`, and `Ethereum Node Records (ENR)`.
- The *IIoT layer* relies on MQTT-based communication using lightweight messages such as `PUBLISH` and `SUBSCRIBE`.

Data collection and feature extraction

In the second step, we collect data and extract the features explained below. The simulator captures all node-level communication in structured JSONL logs, detailing message types, timestamps, latency, energy usage, and message sizes. These logs provide a basis for both temporal and statistical behavior analysis. Each node's activity is analyzed over a sliding window to generate behavioral features, such as protocol entropy, message burstiness, peer diversity, and ENR similarity. These metrics are considered to highlight anomalous behavior characteristics of Sybil activity [56, 210, 215].

Building upon the indicators discussed in the previous sections, we systematically extract 16 features summarized in Table 7.3, grounded in behavioral indicators observed at the network, transport, and application layers to distinguish Sybil nodes from honest participants. These features are designed around several core Sybil-revealing parameters: network latency, residual energy, ENR similarity, protocol usage patterns, message burstiness, peer communication diversity, and traffic volume. Each of these parameters encapsulates a distinct behavioral fingerprint of Sybil nodes, enabling anomaly detection with greater precision [56, 210, 215].

Nodes engaging in Sybil activity frequently exhibit reduced latency due to virtualization or geographical co-location, reflected in the `avg_latency` and `latency_spike_ratio` features [211]. These nodes are also resource-constrained or simulated with artificially fluctuating power levels, making `avg_energy` and `energy_variance` strong indicators of Sybil behavior [216, 217]. Further, cloned Sybils tend to share similar ENR configurations (e.g., IP range, key length, supported protocols), producing high cosine similarity across ENR vectors. This is captured in the `avg_enr_similarity` feature [218, 219].

Protocol-level behaviors also differentiate Sybil nodes. Malicious entities tend to exhibit biased or repetitive use of specific protocol messages, such as frequent `MQTT:PUBLISH` or `discovery:FindNode` transmissions. This imbalance is captured by features such as `mqtt_ratio`, `protocol_entropy`, `dominant_protocol_ratio`, and `discovery_ratio`. Honest nodes generally demonstrate a balanced protocol use pattern, leading to higher entropy and diversity scores.

Temporal traffic analysis further reveals anomalies. Sybil nodes often send messages in rapid bursts, captured via `message_burstiness`, while their interactions are limited to a subset of peers—unlike honest nodes who communicate with a broader set. This is expressed via `unique_peers` and `peer_overlap_ratio`. Therefore, traffic volume is a critical parameter. Sybil nodes may produce either excessive messages (flooding) or abnormally low counts (evasion), depending on their behavioral strategy. These are quantified via `sent_total` and

`received_total`.

Our simulator runs 15 simulation rounds over a network of 300 nodes, with each node transmitting 10 messages per round. This results in 11,107 labeled windows, each representing a snapshot of node behavior across 7 consecutive messages. To capture the temporal dynamics of node behavior, we apply overlapping sliding windows of size 7, where each window summarizes features from 7 consecutive sent or received messages, enabling fine-grained detection of anomalous patterns over time.

In our simulation, the window size indicates how many consecutive messages, including sent or received messages, are grouped together for feature extraction and behavioral analysis. Choosing an appropriate window size is crucial to capture sufficient behavioral context without introducing excessive smoothing or overfitting.

We empirically tested multiple window sizes, including 5, 7, 10, and 12. A small window size (e.g., 5) resulted in underfitting, where limited temporal context led to noisy or unstable features, producing lower accuracy. Conversely, large window sizes (e.g., 10 or 12) resulted in unnaturally high accuracy scores, which we identified as signs of overfitting, due to feature redundancy or capturing patterns too specific to the simulation settings.

A window size of 7 provided a balanced trade-off. It was large enough to capture meaningful behavioral trends and statistical features (e.g., burstiness, protocol entropy, and peer diversity), but small enough to preserve temporal variability and generalize between different nodes. This setting consistently yielded realistic and stable performance across both FL and DFL models, making it the most appropriate choice for our detection framework.

To more accurately reflect adversarial heterogeneity, Sybil nodes are stratified into **high-risk** and **low-risk** categories based on their behavioral patterns and ENR configurations.

- **High-risk Sybil nodes** are designed to exhibit extreme anomalous behaviors such as ultra-low latency, low and unstable energy levels, and high ENR similarity, often resulting from cloned or co-located identities. These nodes typically engage in aggressive tactics like protocol flooding, rapid message spamming, or evasion via ENR manipulation.
- **Low-risk Sybil nodes**, by contrast, are configured to mimic more benign characteristics, with moderate latency, medium energy levels, and ENR similarity. While still adversarial, these nodes blend more closely with honest node behavior, simulating stealthier attack patterns.
- **Honest nodes** maintain high latency, near-maximum energy reserves, and very low

ENR similarity, reflecting stable and diverse communication behaviors.

Dataset preparation:

The extracted features are compiled into a unified CSV file, where each record is labeled based on observed behavior severity (`honest`, `low`, or `high`) and encoded into one-hot vectors to support supervised learning and severity-aware detection. The dataset is then shuffled, grouped by node, and partitioned across clients to support the training step in the detection unit. To ensure balanced representation, the data is normalized and equally distributed per client.

7.4.4 Detection based on Decentralized Federated Learning

Based on the dataset generated in the simulation phase, each node proceeds to train a local model for Sybil detection using decentralized federated learning. The pipeline of our proposed approach is outlined below.

Message collection

MQTT brokers monitor and log all inbound/outbound messages, covering: MQTT operations (e.g., `PUBLISH`, `SUBSCRIBE`, etc.), Ethereum discovery protocols (`ENR:Request`, `ENR:Response`), Gossip (`libp2p`), and control (`devp2p`) messages. Each broker builds a behavioral profile for every node it communicates with.

Local ML Training

Convolutional Neural Networks (CNNs) have been widely adopted in anomaly detection tasks due to their ability to effectively capture local and temporal dependencies in structured data with high computational efficiency [93, 220]. In our model, each participating IIoT broker employs a CNN to perform local Sybil attack detection. Communication features are extracted from protocol logs, including metrics such as entropy, message burstiness, and peer overlap ratios, and used to classify or flag suspicious behaviors. Each broker trains its model locally using a balanced dataset, which is partitioned into training, validation, and testing subsets.

Hyperparameter optimization is conducted using a grid search strategy to identify the most effective configuration. Parameters tuned include the model type (CNN, Random Forest

Table 7.3 Behavioral Features Extracted for Sybil Detection

Feature	Description
sent_total	Number of messages sent in the window
received_total	Number of messages received in the window
protocol_diversity	Number of distinct protocols used
message_burstiness	Std. deviation of inter-message timestamps
mqtt_ratio	Ratio of MQTT messages to all messages
discovery_ratio	Ratio of discovery protocol messages
protocol_entropy	Entropy of protocol usage distribution
unique_peers	Number of unique peers communicated with
dominant_protocol_ratio	Proportion of messages using the most frequent protocol
avg_latency	Mean latency of messages
latency_spike_ratio	Proportion of messages exceeding latency threshold
avg_msg_size	Average message size in bytes
avg_energy	Average remaining energy level
energy_variance	Variance of energy levels
avg_enr_similarity	Cosine similarity between ENRs of node and sampled peers
peer_overlap_ratio	Ratio of repeated peer interactions in the window

Classification (RFC), and Multilayer Perception (MLP)), learning rate, loss function, activation function, kernel size, number of convolutional layers, and dropout rate.

The final CNN architecture consists of three sequential one-dimensional convolutional layers (kernel size 5, padding 2), followed by the Gaussian Error Linear Unit (GELU) activations and batch normalization layers to enhance convergence stability and reduce internal covariate shift. GELU provides a smooth, non-monotonic activation that outperforms traditional functions in deep architectures [221]. Unlike the Rectified Linear Unit (ReLU), which zeroes all negative inputs, GELU retains small negative values probabilistically, enhancing both convergence and generalization.

An adaptive average pooling layer condenses temporal dimensions. This is followed by a fully connected feedforward block composed of three dense layers, interleaved with dropout regularization (rate = 0.3) and non-linear activations.

The model is trained using the Adam optimizer [222] with an initial learning rate of 0.005, which is dynamically reduced via a cosine annealing schedule to improve generalization and avoid sharp minima [223]. Class imbalance is addressed using a class-weighted cross-entropy loss. The final input to the CNN consists of 16 engineered features derived from Ethereum-based IIoT network traffic. The complete training architecture is illustrated in Figure 7.4.

Decentralized federated learning

We leverage the DFL framework to detect Sybil attacks in a blockchain-enabled IIoT environment [224]. This hierarchical and privacy-preserving architecture distributes the learning

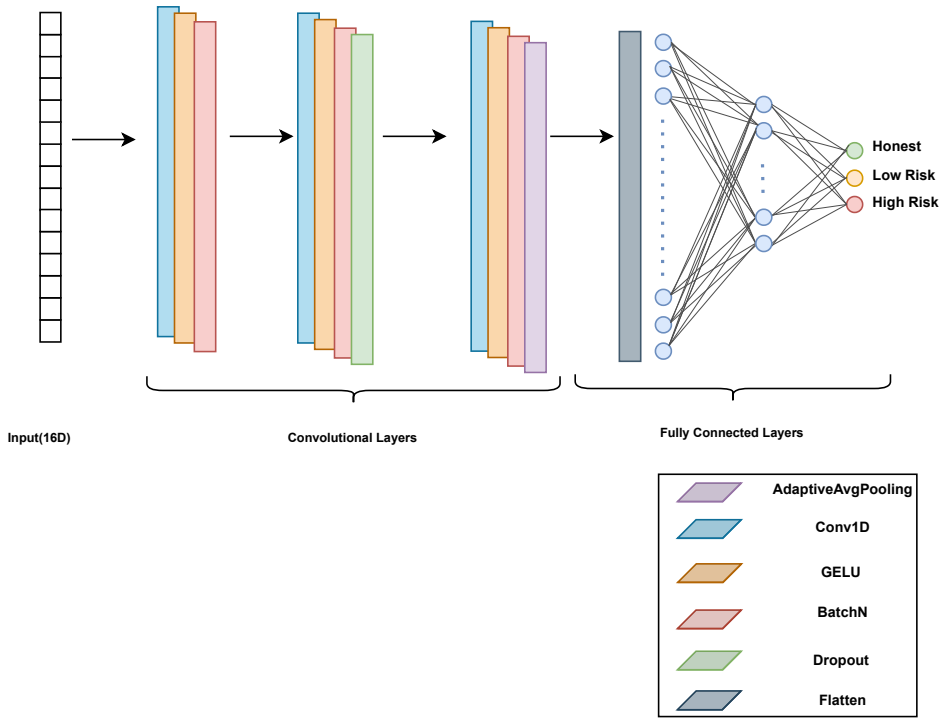


Figure 7.4 The CNN architecture used in our detection model.

process across multiple layers of the IIoT network, effectively minimizing communication overhead while enhancing scalability and detection accuracy.

At the core of the system, each broker i independently trains a local CNN model using behavioral features extracted from observed protocol-level communications. These features include metrics such as entropy, message burstiness, and peer overlap ratios. Upon completing a local training round, each broker shares its model parameters $w_j^{(t)}$ with a set of peers $\mathcal{P}_i$, typically consisting of neighboring MQTT brokers within the same IIoT infrastructure.

To improve generalization and robustness, brokers periodically perform model aggregation in a fully decentralized manner, without relying on a central server. Each broker combines its own model with those received from peers to form a new local model. While classical federated learning typically relies on a central server [225], our system adopts a decentralized, self-inclusive uniform model averaging strategy:

$$w_i^{(t+1)} = \frac{1}{|\mathcal{P}_i| + 1} \left(w_i^{(t)} + \sum_{j \in \mathcal{P}_i} w_j^{(t)} \right) \tag{7.1}$$

where $w_i^{(t)}$ denotes the locally trained model parameters of broker i at round t, and $\mathcal{P}_i$ represents the set of peer brokers from which broker i receives model updates. This formulation ensures an equal contribution from the broker and its peers during aggregation and supports decentralized consensus formation without requiring a centralized aggregator.

This iterative model-exchange process leads to convergence across brokers. As model parameters are repeatedly averaged over multiple rounds, client models tend to stabilize toward a consensus representation. In our implementation, this behavior is confirmed by consistent improvements in validation accuracy and reduction in loss across rounds for all clients. The convergence trend is visualized in the learning curves reported in Section 7.5, demonstrating the practical effectiveness of the proposed decentralized aggregation scheme.

To ensure accountability and coordination, a blockchain layer is integrated to manage the global reputation of participating brokers. Smart contracts maintain tamper-proof records of severity reports and dynamically update reputation scores. In this design, severity scores represent local assessments of Sybil behavior and are submitted to a blockchain-based reputation contract. This contract reduces the global reputation of a node based on the severity of the report. The resulting reputation score reflects the long-term trustworthiness of brokers and is used to enforce adaptive responses such as blacklisting. These scores are forwarded to the reputation management system, as described in the next section. This integration of federated learning with blockchain-based reputation management enables collaborative real-time Sybil prevention while preserving the data privacy of peers.

7.4.5 Prevention Phase via Reputation Management

To ensure robust prevention of Sybil attacks following the detection phase, we introduce an advanced reputation management mechanism that integrates tightly with severity scores produced by the DFL process. Following each federated round, nodes exhibiting anomalies—such as excessive peer overlap, misuse of discovery protocols, or low communication entropy, are flagged by local models and encapsulated into digitally signed reports, each containing the identifier of the suspected node along with a severity score indicating the confidence or intensity of the detected anomaly.

The proportion of instances classified as Sybil is scaled to a discrete 0–10 severity level, which is included in a digitally signed report. In doing so, trust penalties are directly grounded in the behavior learned by each node's model, reinforcing the integrity of the detection process. The reports are submitted to an on-chain reputation management smart contract, which validates each submission using ECDSA recovery to ensure authenticity. To safeguard against false reporting, only authorized validators are permitted to submit anomaly reports. Upon

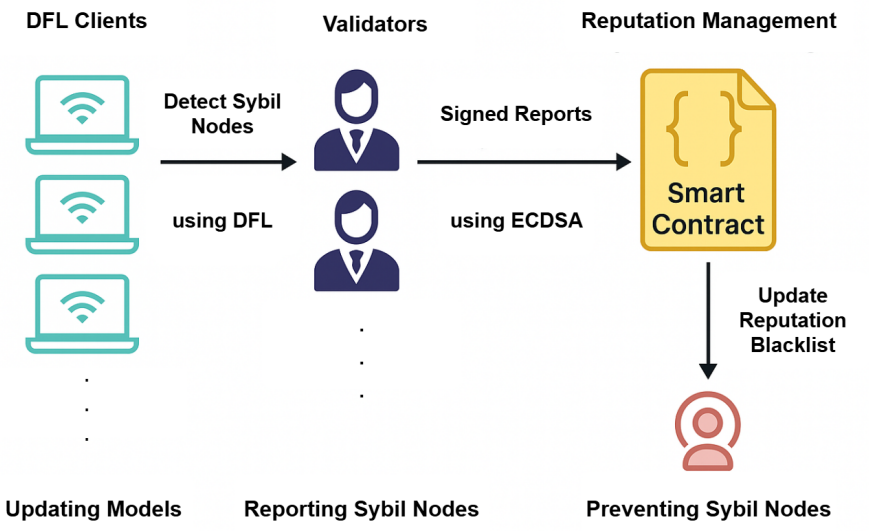


Figure 7.5 The prevention phase using reputation management.

verification, the contract dynamically adjusts the reported node's reputation score based on the severity of the anomaly. If a node's trust score falls below a configurable threshold, the contract automatically blacklists the node, effectively excluding it from subsequent participation in the federated learning or communication processes.

To enhance transparency and model accountability, each node also logs a cryptographic hash of its local model at the end of the training. This traceability feature ensures that nodes can be held accountable for their learned behavior, without incurring the significant gas costs associated with storing full model parameters on-chain. The reputation smart contract maintains a trust score for each node and adjusts it dynamically based on digitally signed severity reports. It validates report authenticity via ECDSA recovery, blacklists nodes falling below a configurable threshold, and logs model hashes for transparency and accountability. The contract exposes key events and admin functions for managing authorized reporters and blacklisted nodes.

Furthermore, a decentralized application (DApp) interface complements the on-chain contract by enabling real-time monitoring, visualization of reputation scores, and administrator-level control actions. Through this interface, operators can monitor anomaly reports, manage blacklisted nodes, and take corrective actions such as issuing warnings or isolating suspicious brokers.

Through this multi-layered architecture, the proposed DFL-driven framework not only detects Sybil behavior with high accuracy but also enforces preventative countermeasures in a secure, scalable, and privacy-preserving manner, making it particularly suitable for sensitive and large-scale IIoT environments. This prevention phase, illustrated in Figure 7.5, operates by Algorithm 20, which outlines the workflow of our smart contract. It verifies validator signatures using ECDSA, applies severity-based penalties to trust scores, and dynamically blacklists malicious nodes. A configurable threshold governs the blacklisting policy, while optional model hash logging ensures transparency and accountability.

7.5 Experimental Results

To implement the proposed Sybil defense approach, we simulate a PoA-MQTT-based IIoT network and monitor communications across both MQTT and Ethereum-related protocols, including ENR, discovery, libp2p, and devp2p. Following data collection, we extract key features and apply dataset balancing techniques to construct a comprehensive dataset for Sybil attack detection. Figure 7.7 presents the t-SNE visualization of the dataset, highlighting the distribution of Sybil and honest nodes. It exhibits distinct clustering among honest, low-severity, and high-severity nodes, indicating that the features extracted from communication patterns provide discriminative power for Sybil detection. We recorded both training and validation loss and accuracy across 100 epochs to evaluate the convergence behavior of the decentralized learning process. The resulting learning curves for each client, capturing both loss and accuracy dynamics, are presented in Figure 7.6. These plots demonstrate consistent improvement and eventual stabilization across all clients, indicating effective convergence of local models through iterative p2p aggregation.

Each client shows a characteristic learning trajectory, with training loss decreasing steadily and validation accuracy improving over time. For instance, client 3 and client 5 exhibit rapid convergence, reaching validation accuracies above 90% within 40 epochs, while client 1 and client 2 show a more gradual but stable improvement. Across all clients, training accuracy continues to rise while validation loss plateaus after approximately 60 epochs, suggesting the model generalizes well without severe overfitting. The separation between training and validation curves remains modest, highlighting the robustness of the DFL framework even in the absence of centralized coordination.

These empirical results support the hypothesis that decentralized FL can achieve consistent and stable model convergence across heterogeneous clients. The consistent reduction in loss and improvement in accuracy validate the effectiveness of peer-based model aggregation in our PoA-MQTT-based IIoT simulation environment.

Algorithm 2 Reputation-Driven Sybil Prevention via Smart Contract

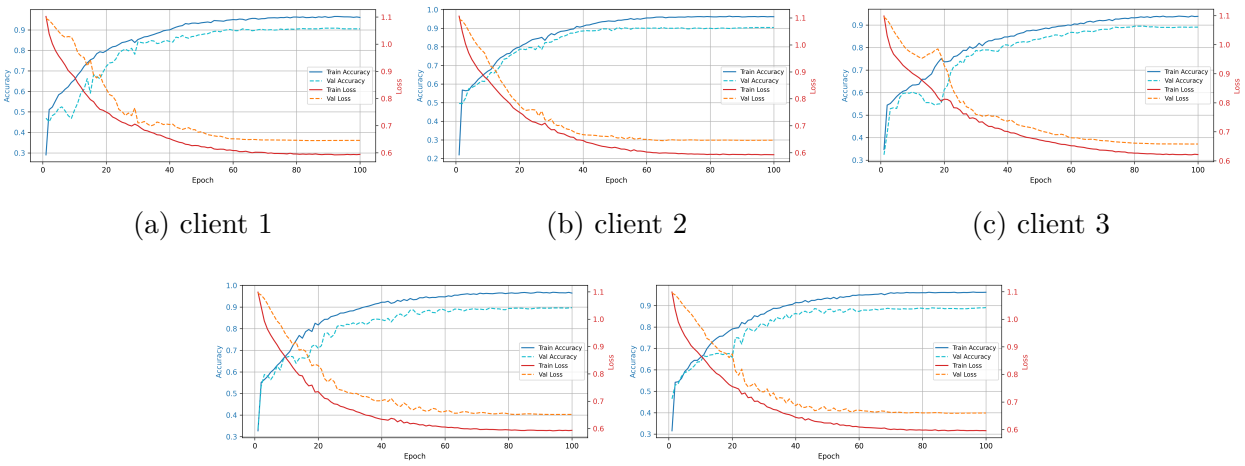
Require: Signed report $\langle node\_id, severity, signature \rangle$
Require: Authorized validators $\mathcal{V}$
Require: Reputation mapping $R[node\_id]$
Require: Blacklist threshold θ
Ensure: Updated trust scores and blacklist status

function ProcessReport(node_id, severity, signature)
    signer ← ECDSA_Recover(node_id, severity, signature)
    if signer $\notin \mathcal{V}$ **then**
        return "Unauthorized validator"
    end if
    rep ← R[node_id]
    if rep.trust_score = 0 **and** rep.num_reports = 0 **then**
        rep.trust_score ← 100 ▷ Initialize on first report
    end if
    rep.num_reports ← rep.num_reports + 1
    rep.trust_score ← max(0, rep.trust_score− severity_to_penalty(severity))
    rep.last_update ← current_block_time
    if rep.trust_score $\leq \theta$ **and** not rep.blacklisted **then**
        rep.blacklisted ← **true**
        Emit `Blacklisted(node_id)`
    end if
end function
function severity_to_penalty(severity)
    return severity
end function

To promote reproducibility and support future research, we publish our Sybil attack simulator alongside the curated dataset, comprising 16 critical behavioral and protocol-level features, via our GitHub repository. We conducted a comprehensive evaluation to compare the performance of DFL against centralized deep learning (DL) and traditional FL using a unified test dataset. The comparison encompasses both evaluation metrics and functional capabilities, as summarized in Tables 7.5 and 7.4, respectively.

Functionally, DFL demonstrates clear advantages over DL and FL in several critical aspects, including data privacy, resilience to SPF, Sybil attack resistance, model tamper protection, and trust scalability. While DFL incurs higher computation and communication overhead compared to DL and FL, this increase in complexity represents a reasonable tradeoff for the enhanced security and privacy guarantees-attributes that are essential in sensitive environments such as IIoT systems.

From a performance perspective, as reported in Table 7.5, all three approaches—DL, FL,



(a) client 1 (b) client 2 (c) client 3

(d) client 4 (e) client 5

Figure 7.6 Learning curves (accuracy and loss) for 5 clients during the 100 epochs.

and DFL—demonstrate strong results across key evaluation metrics, including accuracy, precision, recall, and F1-score. As expected, DL attains the highest overall performance, with an accuracy of 93.46% and an F1-score of 93.48%, benefiting from centralized training and global data access. However, a closer examination of distributed methods reveals that DFL consistently outperforms FL across all clients. Notably, DFL clients exhibit higher and more uniform precision and recall values. For example, DFL Client 2 achieves 92.77% precision and 90.59% recall, resulting in a strong F1-score of 90.86%. In contrast, FL Client 2 reports significantly lower metrics, with 86.88% precision, 87.00% recall, and an F1-score of 86.93%. This pattern holds across all clients: while FL client F1-scores range from 86.55% to 89.43%, DFL client scores range from 89.25% to 91.52%.

These results highlight the advantage of DFL in maintaining both high accuracy and balanced precision-recall tradeoffs across decentralized nodes. Although DL achieves superior centralized performance, DFL offers a practical and scalable alternative for trust-sensitive environments, achieving strong predictive performance without sacrificing privacy or resilience.

Although the differences are not drastic, DFL stands out not only for its predictive performance but also for its architectural and security benefits. Unlike DL and FL, DFL eliminates the need for a central server, enhances data privacy by retaining data locally, and improves resilience against Sybil attacks. These advantages make DFL particularly suitable for trust-sensitive, decentralized environments such as IIoT networks, offering strong performance along with system-level robustness.

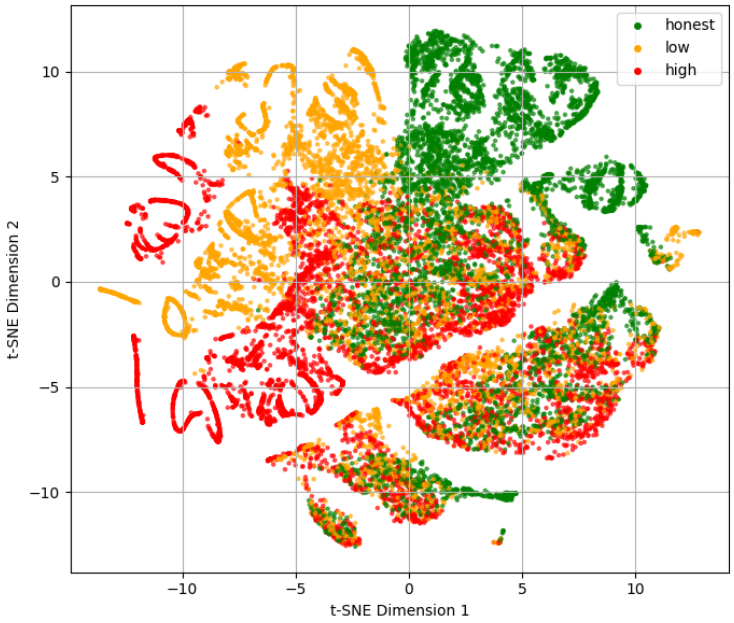


Figure 7.7 t-SNE diagram showing the 2D projection of node behavior features

Table 7.4 Comparative analysis of DL, FL, and DFL across key performance criteria. *Low*, *Limited*, and *Moderate* denote minimal, restricted, and intermediate capabilities, respectively; *High* indicates strong capability. Criteria definitions are detailed in the text.

Criteria	**DL**	**FL**	**DFL**
Data Privacy	No	Partially	Yes
No Single Point of Failure	No	No	Yes
Sybil Attack Resistance	Low	Low	High
Model Tamper Protection	Low	Low	High
Trust Scalability	Low	Limited	High
Time complexity	Low	Moderate	High
Communication overhead	Low	Moderate	High

7.6 Conclusion

In this paper, we propose a defense mechanism against Sybil attacks by integrating DFL with a secure smart contract-based reputation system for both detection and prevention. Our approach begins with the simulation of a PoA-Ethereum-based IIoT network, through which we generate a comprehensive dataset capturing the behavioral patterns of Sybil nodes. Within our DFL framework, each node trains a local model independently, exchanges model weights with neighboring nodes for aggregation, and performs anomaly analysis to detect suspicious behaviors indicative of Sybil activity.

Upon identifying potential Sybil nodes, each participant submits signed severity reports to a

Table 7.5 Comparing evaluation metrics across DL, FL, and DFL (including per-client results)

Metrics (%)	DL	FL Clients					DFL Clients				
		Client 1	Client 2	Client 3	Client 4	Client 5	Client 1	Client 2	Client 3	Client 4	Client 5
Accuracy	93.46	87.08	86.98	86.79	89.30	86.62	89.57	90.61	91.40	90.60	89.93
Precision	93.51	86.94	86.88	86.94	89.44	86.55	91.74	92.77	92.77	92.76	91.58
Recall	93.46	86.97	87.00	86.67	89.42	86.62	89.80	90.59	91.63	90.31	89.11
F1-score	93.48	86.95	86.93	86.67	89.43	86.55	89.77	90.86	91.52	90.71	89.25

smart contract, which then updates trust scores, enforces blacklisting policies when necessary, and optionally records model hashes to ensure verifiability and traceability of model behavior. This tightly integrated architecture of DFL and smart contracts facilitates an adaptive and trust-aware prevention mechanism while maintaining auditable and tamper-resistant records on-chain.

We compare the performance of our DFL-based detection approach with traditional DL and centralized FL techniques. Experimental results demonstrate that DFL achieves competitive detection accuracy while preserving data privacy, as individual clients are not required to share their raw data.

In the future, we plan to enhance our simulator to support additional communication protocols and enable the simulation of other large-scale attacks such as selfish mining. Furthermore, we aim to deploy our framework on a real private PoA Ethereum network and collect real-world IIoT data to validate its effectiveness in practical scenarios.

CHAPTER 8 ARTICLE 5: ADVANCED SMART CONTRACT VULNERABILITY DETECTION USING LARGE LANGUAGE MODELS

Fatemeh Erfan, Mohammad Yahyatabar, Martine Bellaiche, Talal Halabi

Published on IEEE 8th Cyber Security in Networking Conference (CSNet), Paris, France, pp. 289–296, December 4, 2024

Abstract

With the rapid expansion of using smart contracts, protecting the security of these contracts has become crucial. Existing analysis tools for detecting vulnerabilities in smart contracts are unreliable as they often fall short in accuracy, primarily due to their low recall rates—a significant challenge in this field. In this work, we utilize the open-source SolidiFi benchmark dataset to detect vulnerabilities related to Integer overflow/underflow (IoU), reentrancy (RE), and timestamp dependency (TD). These contracts, verified and available on Etherscan, proved unsuitable for direct application of LLMs due to comments, functions, and variables that might reveal the nature of the vulnerabilities. To address this, we performed several preprocessing steps to prepare the dataset for further research. We utilize a large language model to identify vulnerable code, provide reasoning for the vulnerabilities, explain how an attacker might exploit them, and propose fixed code. We design our prompts using chain-of-thought and expert patterns. Finally, we evaluate the results using various metrics and expert reviewers to assess the correctness of the reasoning, potential security risks, and code fixes. Our experiments demonstrate that our approach outperforms existing tools and methods. Notably, our recall rates are significantly high—93.5%, 95.4%, and 93.8%—addressing the challenge of low recall in detecting IoU, RE, and TD vulnerabilities, respectively.

Keywords: smart contract vulnerability, large language model, GPT, security detection, Ethereum, solidity code analysis

8.1 Introduction

With the emergence of Ethereum [178] and the widespread adoption of smart contracts [226] across several domains, various security challenges have also arisen [60]. A smart contract is an executed code, with the result kept as a transaction and stored on the blockchain. Due to the prominent features of blockchain, such as transparency, every node participating in the blockchain can execute the code. Although the processes of storing data in the blockchain and the blocks themselves are immutable, the state of the variables within the smart contract's code is not [19]. An attacker can use these vulnerabilities to launch malicious attacks and

compromise the security or availability of the network if they manage to run vulnerable code. Thus, it is essential to analyze smart contracts from a security and safety perspective.

In recent years, several attacks have been launched on various blockchains, including Ethereum, and have resulted in millions of dollars in losses. Attackers have caused losses totaling approximately $7.69 billion as of March 2024 [129]. Ten critical vulnerabilities were recently reported, including Timestamp Dependency (TD), Integer Overflow/Underflow (IoU), Reentrancy (RE), and others [227]. Although numerous tools have been developed to detect vulnerabilities in smart contracts, such as Slither [72], SmartCheck [71] Oyente [74], HoneyBadger [228], Osiris [75], and Mythril [73], they are not yet sufficiently accurate to be relied upon [126, 128]. According to [126], 80% of smart contract vulnerabilities exploited by attackers were not detected using existing tools. In addition, the evaluations of existing tools may not be fully accurate because the developers used different datasets and evaluation criteria, which may result in biased findings [59].

As a result, developing advanced methods to enhance vulnerability detection in smart contracts becomes imperative; this is where the integration of large language models (LLM) and blockchain security comes into play. Utilizing LLMs' capabilities in natural language comprehension and source code analysis can greatly enhance the detection and mitigation of vulnerabilities in smart contracts. This presents a promising direction for future research and development in the field of blockchain security.

Most existing analysis tools for smart contract vulnerability detection, including LLM-based approaches, suffer from high False Positive (FP) rates, rendering them unreliable for smart contract auditing. Additionally, some tools are designed for older versions of Solidity and are thus incompatible with newer versions of smart contracts. Another challenge in smart contract analysis is the lack of publicly available datasets. Current approaches often concentrate on identifying vulnerabilities without sufficient emphasis on understanding their implications or providing actionable solutions to improve smart contract security. Moreover, the lack of reliable ground truth for smart contract vulnerabilities makes accurate analysis controversial. Addressing these issues is crucial as the rapid growth in smart contract deployments demands faster and more effective auditing processes. Conventional manual methods struggle to handle the volume and complexity of these contracts, often resulting in missing vulnerabilities. By leveraging Generative Pre-trained Transformers (GPT), we aim to quickly analyze and review large volumes of code, ensuring timely and accurate vulnerability detection. These challenges drive us to propose an accurate approach for detecting, explaining, and fixing vulnerable lines of code in smart contracts. Our contributions are as follows:

- We leverage the SolidiFi dataset [170] to provide an open-source and clean dataset of

smart contracts, which is published on our GitHub repository[1]. The original SolidiFi dataset includes comments that indicate the lines of bugs and the types of vulnerabilities, injecting bias into language models for detection with high but unreliable accuracy. Therefore, we have curated a modified dataset in our GitHub repository that is more suitable for further research on smart contract vulnerabilities.

- We utilize GPT-4omni (also known as GPT-4o) to audit smart contracts, aiming to detect three major vulnerabilities more accurately than existing methods. Most current LLM-based works and analysis tools suffer from high FP rates. Instead of relying exclusively on LLMs pre-existing knowledge—which can result in numerous FPs, we employ the chain-of-thought (CoT) approach and expert patterns to design our prompts carefully.
- Most existing approaches categorize smart contracts as either vulnerable or safe, but identifying the specific lines of code exposed to vulnerabilities is crucial. Our method focuses on pinpointing these exact vulnerable lines of code. Additionally, our approach explains the reasoning behind its detection and clarifies why these lines are vulnerable to specific vulnerabilities.
- Finally, our approach proposes fixed code to prevent the smart contract from being exposed to these attacks. It also demonstrates how an attacker could exploit these vulnerabilities, aiding smart contract auditors in detecting and generating secure smart contracts.

The remainder of the paper is organized as follows. We provide important information about smart contracts, vulnerabilities, and LLMs in Section 8.2. Then, we analyze existing works in Section 8.3. Section 8.4 presents our proposed approach. Section 8.5 illustrates our experimental results. Finally, Section 8.6 concludes the paper and outlines future directions.

8.2 Background Concepts

The section provides information on Ethereum smart contracts, LLM, and most critical vulnerabilities.

8.2.1 Smart contract

The idea of a smart contract as a computerized transaction was introduced for the first time by Szabo [58] in 1997. Smart contracts are automated business applications that are self-

[1] `https://github.com/erfan38/SC_vulnerability_detection`

enforceable pieces of code. They typically consist of digital agreements with defined rules and operate without third parties. The creation of smart contracts involves collaboration among multiple blockchain users. After deployment, these contracts are uploaded to the blockchain network and broadcast to all verification nodes [59]. The life cycle of smart contracts consists of four main phases: creation, deployment, execution, and completion.

In the creation phase, parties negotiate the terms, and lawyers draft the initial agreement, which software engineers convert into a smart contract. During deployment, the validated contract is uploaded to a blockchain platform where it becomes immutable, and related digital assets are frozen. The execution phase involves the automatic triggering and execution of contractual clauses once conditions are met, with transactions being validated by blockchain nodes. Finally, in the completion phase, the contract's execution results in updated states and the transfer of digital assets, unlocking them for the involved parties. Throughout these phases, transactions corresponding to contract statements are recorded on the blockchain to ensure transparency and immutability [60]. In all phases, a sequence of transactions is executed and stored on the blockchain.

8.2.2 Smart Contract Vulnerabilities

Blockchain technology has strong security protection but faces vulnerabilities that might compromise its integrity and operation. Recently, a surge in security vulnerabilities in smart contracts, such as the reentrancy attack on *Cream.Finance* [61] that resulted in over $130 million in losses has led to significant financial damage [62]. Several security threats in blockchain systems are crucial vulnerabilities in decentralized finance (DeFi) protocols, supporting services, consensus mechanisms, and smart contracts.

DeFi protocols are exposed to vulnerabilities such as flash loan attacks and Oracle manipulation, which can affect other interconnected protocols within the DeFi space [63]. Supporting services include wallets, exchanges, oracles, and decentralized applications (DApps). Exploiting these services can lead to significant breaches, causing loss of funds, unauthorized data access, and transaction manipulation, with effects spreading across the interconnected ecosystem.

Consensus-based attacks exploit vulnerabilities in consensus algorithms. For instance, double-spending [64], eclipse [52], selfish mining attacks [54], and so on not only undermine blockchain integrity and availability but also gradually reduce trust among its participants [65]. Some existing works [66–68] propose solutions to detect these attacks.

Smart contract vulnerabilities pose substantial risks to blockchain security, as attackers can

exploit these weaknesses to compromise the system. This paper focuses on the most prominent vulnerabilities in smart contracts within blockchain systems, as explained below.

Integer overflow/underflow (IoU)

Integer overflow and underflow vulnerabilities are among the most common bugs in smart contracts. Integer overflow happens when a value is increased in a variable with a limited size until it exceeds its maximum capacity. At this point, the variable's value wraps around to the minimum possible value. For example, if an 8-bit integer can hold values from 0 to 255, adding 1 to 255 would cause an overflow, wrapping the value around 0. Integer underflow happens when an integer value goes below the minimum limit and can be stored in a given variable size. If the same 8-bit integer is at its minimum value (0) and you subtract 1, it wraps around to the maximum limit of 255. In terms of patterns, three sub-patterns exist for Integer *overflow/underflow* vulnerability.

- *ArithmeticOperation (AO):* functions performing arithmetic operations such as addition, subtraction, multiplication, and so on are vulnerable to *overflow/underflow* issues.
- *SafeLibraryInvocation (SLInv):* Functions that use security library functions, such as *Safe Math*, have a lower chance of being exposed to *overflow/underflow* vulnerabilities.
- *ConditionDeclaration (CD):* Functions that use conditional statements (e.g., assert, require) to check arithmetic operations, ensuring they are verified by conditions that prevent overflow/underflow.

A function is considered suspicious for integer overflow or underflow vulnerability if it meets the combined pattern:

$$AO \wedge (SLInv \vee CD)$$

Reentrancy (RE)

When a smart contract allows an external call to another contract while it still has ongoing operations, reentrancy vulnerability may be caused. To transfer an amount of currency to an external address, *sent*, *call*, and *transfer* functions are used. If the external call is made to a malicious contract, it can exploit this situation by recursively calling the original function before the initial execution is complete. This can lead to unexpected behavior and potentially allow an attacker to drain funds from the contract. Regarding patterns of RE, there are four patterns for detecting *Reentrancy* vulnerability explained as follows:

- *CallValueInvocation (CVI):* Functions that call *call.value* are exposed to reentrancy attacks. This pattern checks if there is a call to *call.value* within the function.
- *BalanceDeduction (BD):* This pattern checks if the user's balance is deducted before the money transfer using *call.value*.
- *ZeroParameter (ZP):* This pattern checks if the *call.value* function is called with a parameter of zero. A function has no reentrancy vulnerability (label=0) if it has a *call.value* invocation with a zero parameter.
- *ModifierConstrain (MC):* This pattern checks if the function is restricted by the *onlyOwner* modifier. If a function has no reentrancy vulnerability, it has *onlyOwner* constrain.

A function is considered vulnerable to *reentrancy* vulnerabilities if it matches the following combined pattern:

$$CVI \wedge BD \wedge ZP \wedge \neg MC$$

Timestamp dependency (TD)

In the Ethereum Virtual Machine (EVM), smart contracts operate with limited information, relying on the block's metadata for details like time. However, miners can set arbitrary timestamps within a block without verification from other nodes, leading to potential manipulation. This vulnerability, known as timestamp dependency, occurs when a smart contract uses block timestamps for critical operations. For example, a lottery smart contract might use *block.timestamp* to generate random numbers to determine the winner. Miners can use this timestamp to ensure a desired outcome and compromise the contract's fairness and integrity. Several patterns exist for smart contract vulnerabilities based on expert pattern extraction [62, 69]. For instance, the following are three patterns for timestamp dependency:

- *TDInvocation (TInv):* Invocation of *block.timestamp* is checked by this pattern.
- *TDAssign (TDA):* One reason that makes a smart contract vulnerable to timestamp dependency is to assign *block.timestamp* to a variable or use it in a condition statement. This pattern looks for these two conditions.
- *TDContaminate (TDC):* This pattern checks if *block.timestamp* contaminates the triggering condition of a critical operation or the return value. If the conditional statement affects the function's return value, the function is labeled as having a timestamp dependency vulnerability (label = 1). In addition, if the conditional statement involves

money operations, the function is labeled as having a timestamp dependency vulnerability (label = 1). On the contrary, if the conditional statement does not affect the return value or money operations, the function is labeled as having no timestamp dependency vulnerability (label = 0).

Therefore, if a function satisfies the combination of these sub-patterns, it is considered potentially vulnerable to *timestamp dependency* vulnerability:

$$TInv \wedge (TDA \vee TDC)$$

Analysis Tools

Analysis tools for vulnerability detection in smart contracts are categorized into static and dynamic analysis tools [70]. Static tools such as SmartCheck [71] and Slither [72] analyze the code to detect vulnerabilities without executing it. Most existing static analysis tools produce high FP and False Negative (FN) rates.

On the other hand, dynamic analysis tools involve executing smart contracts. The goal of these tools, such as Mythril [73], Oyente [74], and Osiris [75], as well as fuzzing tools such as ItyFuzz [76], is to detect vulnerabilities that are not identified in static analysis. These tools observe the behavior of smart contracts during execution. However, dynamic analysis can be a resource and time-consuming. Some analysis tools employ both static and dynamic analysis.

One of the challenges with using analysis tools like Slither is that their configuration needs to be adjusted according to the Solidity version of each smart contract. Evaluating a large number of contracts becomes particularly difficult if their Solidity versions differ, as each configuration must be updated individually. Given the limitations in the accuracy of these tools, there is a need for more precise techniques. This paper addresses these challenges in smart contract auditing by proposing an advanced LLM-enabled approach.

8.2.3 Large language models (LLM)

Recently, Large language models (LLMs) have been used in several fields, including cyber security. The field of Artificial Intelligence (AI) has witnessed substantial progress with the emergence of LLMs, especially in the area of natural language processing (NLP). These models can produce and comprehend text with high fluency and coherence. The transformer architecture, a strong foundation for sequence modeling that has fundamentally changed the area of NLP, is primarily responsible for the transformational power of LLMs [63]. This part

covers the fundamentals of LLMs and their application to blockchain security.

Vulnerability detection standards

Regarding LLMs, three standards exist for detection prompting paradigms, including binary, multi-class, and open-ended prompting. In binary prompting, the output of the detection system is true or false. The system's results are chosen between multiple types of known vulnerabilities in multi-class prompting. In the last type, the system detects any potential vulnerabilities [113].

Enhancement of reasoning process

Chain-of-thoughts [114], cumulative reasoning [115], graph-of-thought [116], and tree-of-thoughts [117] are popular techniques to improve the efficiency of LLMs to provide a complex reasoning process that follows human thoughts. Wei et al. [114] explore the reasoning capability of large language models through a technique called chain of thought prompting. The two main ideas of this approach are the generation of natural language rationales for arithmetic reasoning and the in-context few-shot learning capabilities of large language models. Combining these concepts allows for the creation of chain-of-thought prompting, which improves the model's capacity to handle challenging reasoning problems by giving it prompts for input, intermediate reasoning steps, and output. Evaluations on several benchmarks show that this approach works substantially better than traditional prompting. In addition to making it easier to break down complex issues into smaller, more manageable segments, chain-of-thought prompting is also provided. Our methodology leverages the CoT prompting approach to unlock the model's reasoning capabilities.

Generative pre-trained transformers (GPTs)

Several LLMs emerged in 2023, quickly gaining widespread popularity. Among the most notable ones are OpenAI's ChatGPT [118] and Meta AI's LLaMA [119]. For example, ChatGPT alone boasts a user base of over 180 million [120]. OpenAI's GPT has recently transformed NLP with its advanced transformer architecture. GPT models are trained on extensive text corpora, allowing them to grasp statistical language patterns and predict subsequent words. This extensive training results in highly coherent and contextually relevant content.

This paper shows that a language model can significantly enhance the detection of smart contract vulnerabilities. We utilize GPT-4o, a customized variant of GPT-4, to identify vulnerabilities in smart contracts. By leveraging GPT-4o to detect potential vulnerabilities

with greater precision and efficiency, we aim to improve the security of blockchain-enabled decentralized applications.

8.3 Related Work

Hu et al. [113] introduce a two-stage framework named GPTLens using LLMs for smart contract auditing, where the LLM acts as an auditor and critic to augment vulnerability detection. This approach reduces FPs, but the initial stage still generates many potential vulnerabilities that can overwhelm the system or analysts. Liu et al. [164] present PropertyGPT leveraging LLMs to identify smart contract vulnerabilities by analyzing just 13 smart contracts from the *Smart-Bug Curated* dataset [128]. This method surpasses GPTScan [165], detecting 9 out of 13 vulnerable contracts compared to GPTScan's 5 out of 13. However, the method focuses exclusively on binary detection of limited vulnerabilities. Sun et al. [165] introduce GPTScan utilizing GPT-3.5-turbo with static analysis to identify some smart contract vulnerabilities. However, its effectiveness can be limited by the precision of static analysis, which may not always capture vulnerabilities in larger or more complex smart contracts.

Ince et al. [168] fine-tuned two open-source models, Meta's Code Llama and Instruct models, with 34 billion parameters to detect smart contract vulnerabilities. However, the authors relied on smart contract analysis tools with low detection rates and experienced inconsistent performance across different vulnerabilities. In addition, the large 34 billion parameter models required high-end A100 80 GB GPUs, making them costly and less accessible. Ghaleb et al. [170] introduced SolidiFI, a framework for evaluating existing smart contract bug detection tools. They provide a dataset created by injecting bugs into real-world smart contracts sourced from *Etherscan*[2]. Du et al. [5] evaluate smart contract auditing using GPT-4 on the SolidiFI benchmark and argue that GPT-4 is unsuitable for vulnerability detection. However, their study did not focus on prompt engineering, which impacted the robustness of their results and resulted in a performance inferior to random classification. While their work targets binary classification, our approach integrates reasoning for enhanced detection.

8.4 Methodology

To address the limitations of existing works, we propose an approach for detecting smart contract vulnerabilities using GPT models. Instead of solely relying on GPT's knowledge to identify vulnerabilities, an approach prone to many FPs and limited by GPT's pre-existing

[2] `https://etherscan.io`

knowledge, we leverage the chain of thought approach and expert patterns to carefully design our accurate prompt. We utilize expert patterns in the instruction as well as expert auditors in the few-shot examples part of the chain of thought to 1) specify the location of the vulnerabilities within the code, 2) explain the reasoning behind the detection, 3) how an attacker can exploit the detected vulnerability, and 4) write the correct format of code to prevent such attacks.

The overall architecture of our proposed approach is illustrated in Figure 8.1, which contains four phases: (1) data preparation, which prepares the existing dataset by completing some preprocessing steps; (2) prompt design phase, which makes our LLM model compatible with our objectives and datasets; (3) generation phase, where our model generates the results; and (4) evaluation phase, where we use some metrics to evaluate our proposed approach as well as obtain some reviews from experts. The details of the phases are given as follows.

8.4.1 Data Preparation

The detection model's results on datasets with and without comments should be the same. However, this might not be true when a language model comprehends the code and detects the vulnerability. Specifically, using a dataset that includes comments introduces bias, helping the model better identify vulnerable lines of code. As expected, our approach performs better on the original SolidiFi dataset [170] than on the preprocessed version, as detailed in Section 8.5. The SolidiFi dataset injects bugs into healthy contracts and highlights these changes with comments, function names, and variable names. This discrepancy arises because LLMs learn from context; comments, function names, and variable names provide information that improves detection accuracy. As a result, the findings are not entirely realistic. Therefore, using datasets with comments for large language models may not yield realistic results. We address this issue by preparing a dataset as described below. This phase consists of two sub-phases: data preprocessing and training data. We work with a dataset containing comments, functions, and variables related to vulnerabilities. During data preprocessing, we rename functions and variables associated with vulnerabilities and remove biased comments. Additionally, we update the vulnerability information, including the lines of vulnerable code, accordingly. The prepared dataset is then sent to the next step. Figure 8.2 shows an example of the preprocessing sub-phase.

Auditing is conducted for one smart contract per vulnerability in the training data phase. This involves preparing reasoning for identifying vulnerable code, explaining potential security risks, and providing fixes. The reasoning part explains why specific lines of code are vulnerable. The potential security risk section describes how an attacker might exploit the

Figure 8.1 Main phases of the proposed approach.

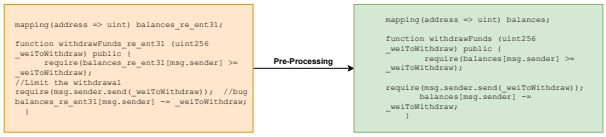


Figure 8.2 The preprocessing step in the data preparation phase.

vulnerability. Finally, in the code fixing part, we replace vulnerable lines of code with secure alternatives. The training data plays the role of Few-shot examples in CoT prompt engineering of the approach.

8.4.2 Prompt Design

This part explains the steps of the prompt design phase as shown in Figure 8.3.

System message

The System Message is a special type of message used to guide the behavior of a language model. As the first message in a chat with the LLM, it defines the model's role and sets the context and boundaries for the interaction. This message instructs the model on responding, including details such as tone, style, persona, and specific task-related instructions. Properly framing this context is essential for the LLM to generate relevant and accurate responses.

Chain of thoughts

In this phase, we use the chain of thoughts approach to design a prompt to detect vulnerabilities, lines of code, potential security risks, and fixed code. The important step in our proposed approach is to design an accurate prompt, increasing our approach's performance. To address this concern, we utilize expert auditors and expert patterns [62, 69]. The prompt includes tasks, instructions, few-shot examples, and data structure alignment. In the *task*, the overall objective of the prompt is defined. It states what goal must be reached by the LLM. In the *instruction* part, clear directions (on how LLM must approach the task) are provided to guide LLM while generating a response. Based on existing expert patterns, we design accurate prompts by defining suitable instructions for *timestamp dependency, reentrancy,* and *integer overflow/underflow* vulnerabilities detection. The *few-shot examples* are sample inputs and outputs provided to the LLM to show the desired format and quality of responses. These few-shot examples provided by expert auditors serve as a reference for the model, guiding it in managing similar tasks that can significantly improve its performance. *Data structure alignment* ensures that the data presented to the LLM is organized so the model can easily process and understand. This includes formatting the input data specifically and using standardized structures to ensure consistency. Proper data structure alignment helps the LLM to generate more accurate and relevant responses by making the input clearer and more manageable. The LLM gets the designed prompt as an input and gives the desired responses. Figure 8.4 illustrates an example of a designed prompt for smart contract vulnerability detection.

8.4.3 Results Generation

In this phase, the designed prompt is provided to the GPT-4o model, with a smart contract as the input. The prompt is designed to generate responses, including lines of code, reasoning, potential security risks, and corrected code. The model's evaluation is formatted as a JSON object, as specified in the prompt. Since some smart contracts can be lengthy, we split the code into smaller sections to avoid exceeding the LLM's token limitations. Our strategy identifies split points using keywords (e.g., 'function') and ensures sections do not exceed a maximum line limit (in this case, 120 lines). The results are combined into a single JSON object, addressing the challenge of processing long smart contracts and token limitations while enhancing the model's accuracy and performance, as many real-world contracts are complex and extensive.

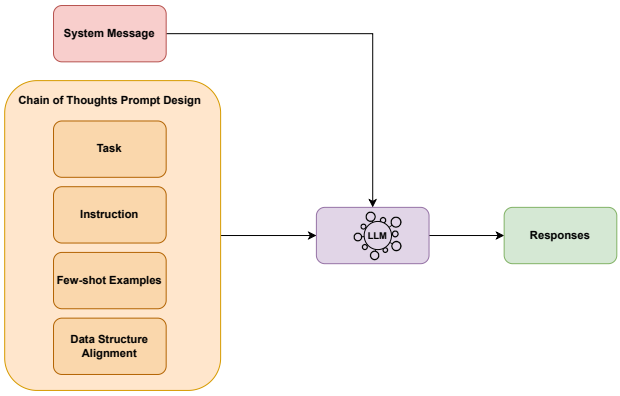


Figure 8.3 The steps of prompt design.

System Message

You are a cyber-security expert who analyses and audits smart contracts and provides reasoning, potential security risks, and fixed code.

Task

In the code below, detect {vulnerability} vulnerabilities through different lines of codes. Go through each line until the end of the code. Provide lines of happening vulnerability (start-end), explain its reason, describe the potential security risk it introduces, and rewrite the code without the vulnerability.

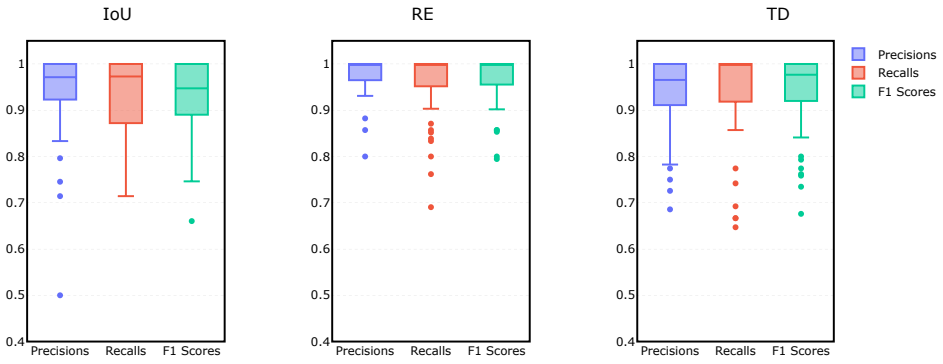
Few-shot Samples

{"vulnerableLines": "<start line>-<end line>",
"vulnerabilityReason": "<Reasoning for the vulnerability>",
"potentialSecurityRisk": "<Potential risks the vulnerability causes>",
"fixedCode": "<the enclosing function with no bugs or vulnerability>"}

Figure 8.4 An example of a prompt to detect smart contract vulnerabilities.

8.4.4 Model Evaluation

Our model is evaluated using standard metrics such as precision, recall, and F1 score in this phase. We also leverage expert reviewers to review the correctness of the results. Since the LLM is designed to specify the exact vulnerable lines of code, we only consider the results as truly detected if the identified lines fall within the corresponding ground-truth lines of code. For example, the detection is not considered as accurate if the result's lines of code part are $45-49$, but the ground-truth lines are $46-50$. It is important to note that the LLM returns vulnerable lines of code, starting with the line containing the vulnerability and extending to the affected function.



(a) (b) (c)

Figure 8.5 Smart contract vulnerability detection results.

8.5 Experimental Results

Our experiment utilizes a dataset of 150 smart contracts from the SolidiFi dataset [170], which includes a total of 4,057 injected bugs: 1,333 vulnerabilities related to IoU, 1,343 RE, and 1,381 TD issues. These contracts make the generation biased due to their comments, functions, and variables that might reveal vulnerabilities. The dataset used in our experiments (with names altered and comments removed) is available in our GitHub repository.

As previously mentioned, we focus on three types of vulnerabilities: Integer IoU, RE, and TD. We specifically analyzed two GPT models, GPT-4o and GPT-4o mini, on the modified dataset with results summarized in Table 8.1. GPT-4o outperforms GPT-4o mini in terms of precision, recall, and F1 score. However, the APIs for GPT-4o mini are significantly cheaper and faster than those for GPT-4o. Therefore, auditors can choose between them based on their needs, whether prioritizing accuracy or cost and speed. In both models, RE detection results are higher than those for IoU and TD. GPT-4o achieved an F1 score of 96.7% in RE detection, outperforming the 92.8% and 93.6% F1 scores for IoU and TD, respectively. GPT-4o-mini achieved a 94.6% F1 score for RE detection, while its F1 scores for IoU and TD are 89.3% and 90.3%, respectively.

Additionally, we compared our approach to existing analysis tools and a recent paper on the same dataset [5]. The analysis tools were evaluated using the full SolodiFi dataset (50 smart contracts per vulnerability), while our approach was tested on 49 smart contracts (one smart contract was used in input prompts for the LLM). A recent paper [5] tested their approach on only 13 smart contracts. As shown in Table 8.2, our proposed approach achieves lower FP and rates across all three vulnerabilities compared to the other tools.

Table 8.1 Comparison of GPT-4o and GPT-4o mini.

Language Models	GPT-4o			GPT-4o mini		
Vulnerabilities	IoU	RE	TD	IoU	RE	TD
Precision	92.2	98.1	93.4	88.2	94.5	92.2
Recall	93.5	95.4	93.8	90.3	94.8	88.5
F1 Score	92.8	96.7	93.6	89.3	94.6	90.3

Finally, our approach is compared to the functionality outlined in Table 8.3. It excels in detection (identifying vulnerable lines of code), reasoning (explaining the vulnerabilities and potential exploit methods), and code correction (fixing the vulnerable code), whereas [5] does not address code fixing. Other tools offer only binary detection without reasoning or code correction capabilities.

Despite achieving better results with the original dataset, we rely on the results from the modified dataset. This dataset more accurately represents real-world smart contracts, making the results more reliable. Figure 8.5 illustrates the precision, recall, and F1 score distribution for detecting 150 vulnerable smart contracts in the modified dataset. Sub-figures (a), (b), and (c) show the distribution of evaluation metrics for all smart contracts for IoU, RE, and TD, respectively. The box plot analysis indicated that the median recall rates for IoU, RE, and TD are 97.3%, 100%, and 100%, respectively. This means that the majority of contract detections have Recall rates that meet or exceed these values. The precision for the majority of vulnerable smart contracts for the IoU, RE and TD are 97.2%, 100%, and 97.5%, respectively. The F1 score rates for the majority of vulnerable smart contracts detected for the mentioned vulnerabilities are 94.7%, 100%, and 97.5%, respectively.
Common metrics were used to evaluate the lines of code. Since there is ground truth for the lines of code, a quantitative evaluation was established. However, expert reviewers evaluate the three fields of output to assess the reasoning, potential risks, and corrected code. The experts' analysis shows that our proposed approach delivers practical and reliable results for smart contract auditing.

8.6 Conclusion and Future Work

This work utilized the open-source SolidiFi benchmark dataset to identify vulnerable smart contracts with IoU, RE, and TD vulnerabilities. Due to the probable side effects of injecting bugs into the smart contracts, we conducted a preprocessing phase to create an unbiased dataset that is now available in our GitHub repository for further research. GPT-4o and GPT-4o mini models can detect vulnerabilities and provide reasoning, as well as their potential

Table 8.2 Comparison of the number of false positives and false negatives for different approaches on the SolidiFi dataset.

Approaches	Our approach (GPT-4o)*			Oyente [74]			Mythril [73]			SmartCheck [71]			[5]**		
Vulnerabilities	IoU	RE	TD	IoU	RE	TD	IoU	RE	TD	IoU	RE	TD	IoU	RE	TD
False Positive	109	24	90	0	164	495	137	280	0	0	1237	561	3	5	0
False Negative	90	61	84	898	844	886	932	805	810	1072	106	341	89	8	97
# Vulnerable functions	1315	1319	1363	1333	1343	1381	1333	1343	1381	1333	1343	1381	104	104	109

: we selected 49 smart contracts out of 50 (one contract is used as few-shot examples)
**: [5] only selected 13 out of 50 smart contracts

Table 8.3 Functionality comparison

Approaches/ Features	Detection	Reasoning	Code correction
Our approach	✓	✓	✓
Oyente	✓	✗	✗
Mythril	✓	✗	✗
SmartCheck	✓	✗	✗
[5]	✓	✓	✗

security risk and fixed code. Our approach, guided by CoT reasoning and expert patterns, demonstrated superior recall, precision, and F1 score performance compared to existing tools.

For future work, we aim to enhance model consistency and address inherent issues in LLMs, such as hallucinations, by incorporating self-evaluation mechanisms and explainable AI methods. Additionally, we will develop a specialized GPT model designed to detect and rectify vulnerabilities in smart contracts while generating secure contracts. We also intend to create a user-friendly interface to provide developers with real-time feedback.

CHAPTER 9 ARTICLE 6: FINE-TUNED LARGE LANGUAGE MODEL AND COMPREHENSIVE DATASET FOR SECURING ETHEREUM SMART CONTRACTS WITH REAL-TIME VSCODE AUDITING

Fatemeh Erfan, Mohammad Yahtatabar, Martine Bellaiche, Talal Halabi

Submitted to Elsevier Journal of Expert Systems with Applications, June 8, 2025

Graphical Abstract

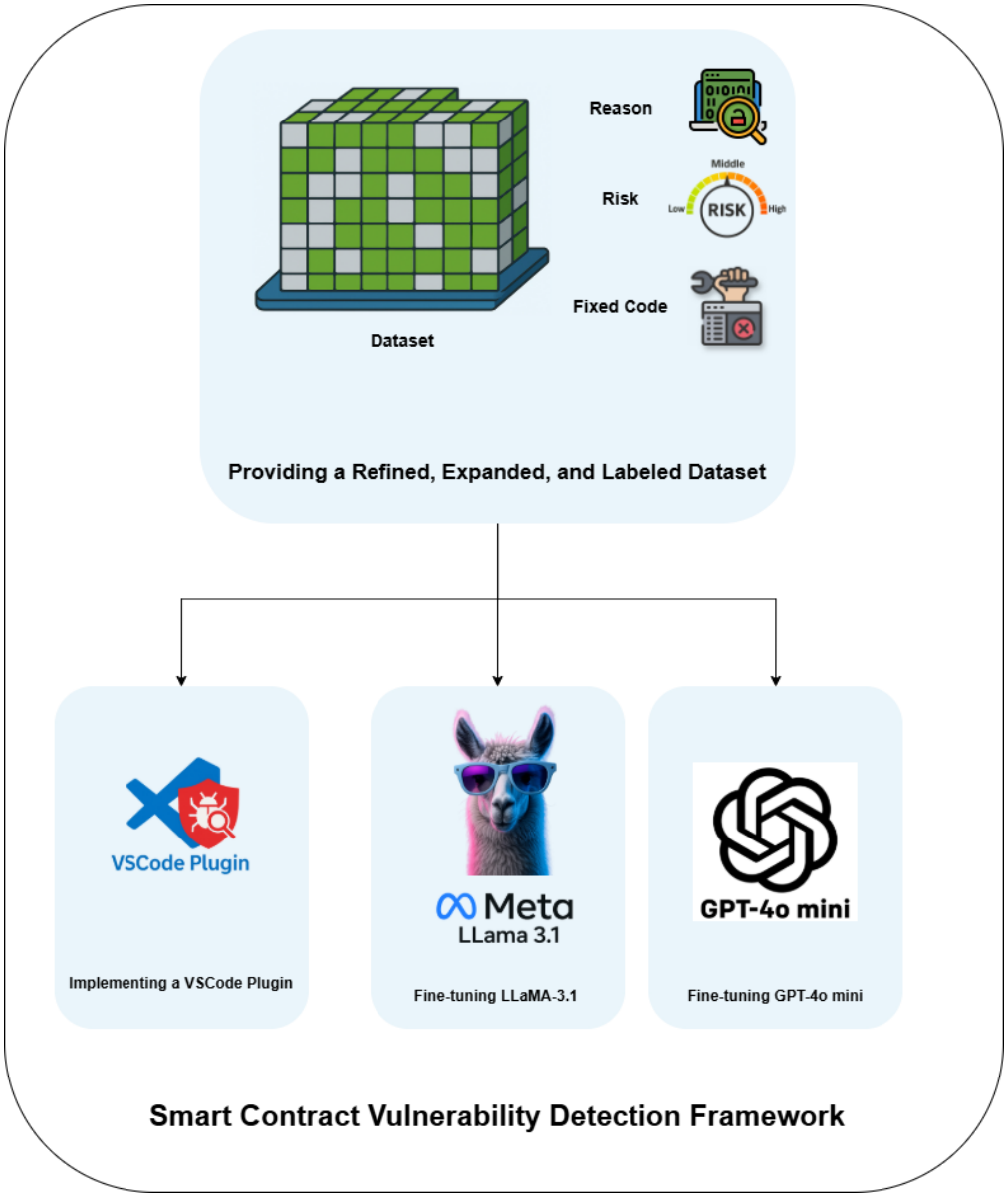


Figure 9.0 Graphical Abstract

Highlights

- Created a refined, expanded, and precisely labeled dataset with explanations, risk assessments, and fixes for each vulnerability
- Fine-tuned an open-source LLM: LLaMA-3.1-8B using parameter-efficient techniques (LoRA) for smart contract vulnerability detection
- Fine-tuned GPT-4o-mini on the same corpus for comparative analysis
- Developed a real-time VSCode plugin integrating GPT-4o-mini for smart contract auditing
- Released the datasets, tool, and fine-tuned model to advance research in smart contract security

Abstract

Since the advent of Ethereum, ensuring the security of smart contracts has become imperative. Integer overflow and underflow, reentrancy, and timestamp dependency remain the three most prevalent vulnerabilities in deployed contracts. Existing static-analysis tools often yield insufficient accuracy, and datasets derived from them inherit the same shortcomings. Moreover, the smart contract ecosystem lacks a dependable, real-time auditing aid for developers and a fine-tuned model trained on a truly comprehensive corpus. In this paper, we present three main contributions. (1) Dataset curation: the state-of-the-art vulnerability datasets are aggregated and harmonized, producing a clean, fully labeled dataset that integrates detailed explanations, potential security risks, vulnerable line ranges, code snippets, and corresponding fixes. The dataset is publicly available via our GitHub repository. (2) Model fine-tuning: the *LLaMA-3.1-8B* model as well as *GPT-4o-mini* are fine-tuned on this corpus and evaluated with both standard classification metrics and text-quality measures. The fine-tuned *LLaMA-3.1* model achieves a precision of 93.55%, an average semantic similarity of 77.48%, and a code similarity of 87.25%. (3) IDE integration: We implement a real-time Visual Studio Code extension, backed by the GPT-4o API, that highlights, explains, and automatically patches vulnerabilities as the developer writes. Together, these contributions deliver a rigorously validated model and a practical developer toolchain that markedly advance the state of smart-contract security research and practice.

Keywords: Smart contract vulnerability; large language model; GPT; vulnerability detection; Ethereum security; solidity code analysis; security dataset

9.1 Introduction

Today, Ethereum is the second most widely used blockchain network [178]. Its popularity has led to the widespread adoption of smart contracts, which in turn has introduced a variety of security challenges [60]. A smart contract is an executable segment of code whose outcomes are recorded as transactions and permanently stored on the blockchain. Due to the transparent nature of blockchain, any participant can execute this code. Despite the blockchain's inherent immutability in data and block storage, the internal state of variables within smart contracts remains mutable during execution [19]. If a smart contract contains a vulnerability, malicious actors can exploit it, thereby posing serious threats to the security and integrity of the blockchain network. Consequently, pre-deployment analysis of smart contracts is essential to mitigate such risks.

In recent years, blockchain platforms such as Ethereum have experienced several high-impact security breaches, resulting in billions of dollars in financial losses. Among the most frequently exploited vulnerabilities are reentrancy (RE), integer overflow/underflow (IoU), and timestamp dependency (TD) [229]. Although several tools such as Oyente [74], Mythril [73], and Slither [72] have been developed to detect these vulnerabilities, they are not fully reliable due to limitations in detection accuracy and consistency. Existing studies have shown that a significant proportion of exploited vulnerabilities remain undetected by these tools [128].

Recent advancements have explored the integration of large language models (LLMs) in the domain of smart contract vulnerability detection, offering promising improvements. However, many of these studies rely on datasets generated from outputs of existing detection tools. Given the shortcomings of those tools, the resulting datasets are inherently noisy, incomplete, and not sufficiently robust for comprehensive vulnerability analysis.

Additionally, there remains a lack of a dedicated and robust model specifically fine-tuned for smart contract vulnerability detection. Most existing models either rely on limited or unstable datasets or lack completeness in scope. Moreover, existing approaches seldom address the remediation aspect of vulnerabilities, leaving an important gap in actionable security solutions.

Another key motivation for this study is the absence of a real-time vulnerability detection tool to assist developers during the process of writing smart contracts. There is a clear need for a reliable, LLM-based tool capable of identifying potential vulnerabilities as the contract is being developed. Furthermore, such a tool should offer clear explanations of the detected issues, describe the potential security implications and attack scenarios, and most importantly, provide practical guidance for code correction. To the best of our knowledge,

no existing tool currently integrates this comprehensive functionality.

This paper contributes to blockchain security by addressing the aforementioned limitations through the following:

- **Creating a comprehensive dataset for smart contract vulnerability detection.** We improve and extend an existing dataset used for smart contract vulnerability detection by refining and expanding vulnerability annotations. This dataset serves as the foundation for fine-tuning an open-source model for securing smart contracts.
- **Fine-Tuning an Open-Source LLM for Smart Contract Security.** The *LLaMA-3.1-8B* was fine-tuned on our improved dataset using *LoRA for memory-efficient training*. This approach enables a cost-effective and scalable deployable alternative to API-based vulnerability detection. We also fine-tune GPT-4o-mini and compare the fine-tuned models with their base LLMs.
- **Developing a VSCode Plugin for Smart Contract Security Analysis.** We implement a *VSCode plugin*, enabling real-time vulnerability detection and remediation assistance using *GPT-4o APIs*. This tool integrates AI-assisted security auditing directly into developers' workflows, providing on-the-fly insights while they write smart contracts. Importantly, this contribution remains *API-based*, leveraging OpenAI's GPT-4o and GPT-4o-mini models for vulnerability detection, similar to our previous work.

By integrating these contributions, we provide both an immediate, practical solution for developers (VSCode Plugin using GPT-4o APIs) and a long-term, self-hosted alternative (fine-tuned LLaMA-3.1-8B-instruct), advancing blockchain security through both accessibility and model independence.

The remainder of the paper is organized as follows. Section 9.2 provides essential information about smart contracts, vulnerabilities, and LLMs. Then, existing works are analyzed in Section 9.3. Section 9.4 presents our proposed approach. Section 9.5 demonstrates our experimental results. Finally, Section 9.6 concludes the paper and outlines future directions.

9.2 Background Concepts

This section provides information on Ethereum smart contracts, LLM, and the most critical vulnerabilities.

9.2.1 Smart Contract

Smart contracts function as self-executing business applications, composed of predefined rules that eliminate the need for intermediaries. These contracts are developed collaboratively by multiple blockchain users and, once deployed, are stored on the blockchain and disseminated to all verification nodes. The smart contract lifecycle comprises four key phases: creation, deployment, execution, and completion.

During the creation phase, stakeholders define the terms of the smart contract, which are first written up by legal professionals and then converted into executable code by software developers. Deployment consists of uploading the finalized contract to a blockchain network where it becomes immutable, and any associated digital assets are locked. When pre-defined conditions are met, this automatically triggers the execution phase, providing an effective manner to enforce the terms of the contract validated through the blockchain process. Finally, the completion phase completes the execution of the contract and updates the states and asset transfer, releasing the locked resources to the involved parties. During these stages, the blockchain transactions are recorded securely, guaranteeing transparency and permanence [60]. All contract-related transactions adhere to that order to maintain the immutability of the contract interactions on the blockchain.

9.2.2 Smart Contract Vulnerabilities

Blockchain technology is built to provide strong security yet remains vulnerable to certain threats that can undermine its reliability and trustworthiness. Recently, an increasing number of security vulnerabilities in smart contracts have caused devastating financial losses. For instance, a recent exploit targeting a RE vulnerability led to damages exceeding $130 million, underscoring the significant risks posed by such security flaws [62]. These threats affect various components of blockchain ecosystems, including decentralized finance (DeFi) protocols, core infrastructure, consensus mechanisms, and smart contracts.

DeFi protocols are particularly susceptible to attacks such as flash loan exploits and Oracle manipulation [63]. Since these financial applications are deeply interconnected, a single vulnerability can have profound implications, affecting multiple platforms [63]. In addition to DeFi, other supporting blockchain services, including decentralized exchanges, digital wallets, oracles, and decentralized applications (DApps), are prime targets for attackers. Security breaches in these areas can lead to unauthorized transactions, loss of assets, and manipulation of critical network data, creating widespread instability.

Consensus-based attacks take advantage of vulnerabilities in consensus mechanisms. Ex-

amples include double-spending, eclipse attacks, and selfish mining attacks, all of which compromise the integrity and availability of blockchain networks while undermining trust among participants [65]. Existing works explored various detection and mitigation strategies [67, 68, 198], yet these threats continue to pose significant challenges. Among all these risks, smart contract vulnerabilities stand out as one of the most pressing concerns. Weaknesses in contract logic have the potential to cause substantial financial losses and compromise the trustworthiness of blockchain-based applications. This study delves into the most pressing security challenges in smart contracts, examining their underlying causes and exploring viable strategies for detection, as outlined below.

Integer Overflow and Underflow in Smart Contracts

Integer overflow and underflow are critical security vulnerabilities in Ethereum-based smart contracts. These issues arise due to limitations in the Ethereum Virtual Machine (EVM) and how Solidity handles integer types. While Solidity provides various integer types, all integers are represented as 256-bit values in the EVM. Prior to Solidity version 0.8.0, overflow and underflow were not automatically checked, leading to unintended behavior. The integer overflow occurs when an arithmetic operation exceeds the maximum value, wrapping around to the minimum. Conversely, integer underflow happens when a value drops below the minimum, wrapping around to the maximum. For example, an 8-bit integer (0–255) overflows from 255 to 0 and underflows from 0 to 255.

Detecting and mitigating these vulnerabilities is crucial for smart contract security. Researchers have identified three key patterns associated with these vulnerabilities:

- *Arithmetic Operation (AO):* Functions involving arithmetic operations such as addition, subtraction, and multiplication may be susceptible to overflow or underflow [71, 230].
- *Safe Library Invocation (SLInv):* Security libraries like *SafeMath* prevent overflow/underflow [71, 230, 231].
- *Condition Declaration (CD):* Explicit checks using `require` or `assert` enforce logical constraints to prevent overflow and underflow [71, 230, 231].

Reentrancy (RE)

Reentrancy vulnerabilities arise when a smart contract makes an external call to another contract before completing its current execution. This issue typically occurs when functions such as *send*, *call*, and *transfer* are used to transfer cryptocurrency. If the external contract

is malicious, it can exploit this mechanism by repeatedly invoking the original function before the initial transaction finalizes. This can lead to unintended consequences, potentially enabling an attacker to deplete the contract's funds.

A typical attack scenario involves an adversary deploying a malicious contract and linking it to a vulnerable contract. By depositing a certain amount, say 1 Ether, and initiating a withdrawal, the victim contract processes the transaction and transfers funds. However, before the contract updates its balance, the attacker's contract recursively calls the withdrawal function. This cycle repeats until either the contract is drained or it reaches a system-imposed limit on computational resources.

To identify reentrancy vulnerabilities, four distinct patterns are usually analyzed [232]:

- *CallValueInvocation (CVI):* This pattern examines whether a function contains a *call.value* operation, as such calls are susceptible to reentrancy.
- *BalanceDeduction (BD):* It verifies whether the contract deducts the user's balance before initiating a fund transfer via *call.value*, which is a key safeguard against reentrancy.
- *ZeroParameter (ZP):* This pattern checks if *call.value* is invoked with a zero argument. If a function includes *call.value(0)*, it is generally not prone to reentrancy (assigned a label of 0).
- *ModifierConstrain (MC):* This pattern evaluates whether a function is protected by the *onlyOwner* modifier, which restricts access to authorized users and mitigates reentrancy risks.

Timestamp Dependency (TD)

One of the most common smart contract vulnerabilities is timestamp dependency. This vulnerability is also known as time manipulation. In Ethereum Virtual Machine (EVM), smart contracts function with restricted access to external data, relying on block metadata for time-related information. However, since miners can assign timestamps within a specific range when creating a block, this flexibility introduces the risk of manipulation. Timestamp dependency arises when a smart contract utilizes block timestamps for crucial operations. For instance, in a lottery contract that generates random numbers using *block.timestamp*, a miner could influence the outcome by adjusting the timestamp, thereby compromising the contract's fairness and security. Several patterns exist for smart contract vulnerabilities based on expert pattern extraction [62, 69]. For instance, the following are three patterns for detecting timestamp dependency [232]:

- *TDInvocation (TInv):* This pattern detects instances where *block.timestamp* is explicitly invoked within the smart contract.
- *TDAssign (TDA):* One common cause of timestamp dependency vulnerabilities is the assignment of *block.timestamp* to a variable or its use within a conditional statement. This pattern is designed to identify occurrences of both cases.
- *TDContaminate (TDC):* This pattern assesses whether *block.timestamp* influences the triggering condition of a critical function or impacts its return value. If the conditional statement determines the function's output, the function is flagged as vulnerable to timestamp dependency (label = 1). Similarly, if the condition involves financial transactions, the function is also marked as vulnerable (label = 1). However, if the conditional statement neither affects the return value nor involves monetary operations, the function is considered free from timestamp dependency issues (label = 0).

Analysis Tools

Smart contract vulnerability detection tools are generally classified into static and dynamic analysis tools. Static analysis tools such as SmartCheck [71] and Slither [72] examine the source code without executing it to identify potential security flaws. However, many static analysis approaches are prone to high false positive (FP) and false negative (FN) rates, affecting their reliability.

In contrast, dynamic analysis tools operate by executing smart contracts to monitor their runtime behavior. Tools such as Mythril [73] and Oyente [74] along with fuzzing-based approaches such as ItyFuzz [76], are designed to detect vulnerabilities that static analysis might overlook. Although dynamic analysis can provide deeper insights into contract behavior, it is often computationally expensive and time-intensive. Some modern tools integrate static and dynamic techniques to improve detection accuracy.

A notable challenge in using tools like Slither is the necessity of configuring them according to the specific Solidity version of each contract. When analyzing a large dataset of contracts with varying Solidity versions, manual adjustments to the tool configuration become a cumbersome and time-consuming task. Given the accuracy limitations of existing tools, there is a clear need for more effective solutions. This study aims to address these challenges by introducing an advanced LLM-based approach to detect vulnerabilities in smart contracts.

9.2.3 Large Language Models (LLM)

The rapid progress of artificial intelligence (AI) has led to groundbreaking developments in various areas, especially cybersecurity. Among these innovations, LLMs have emerged as invaluable tools, particularly in natural language processing (NLP). The ability of LLMs to generate and interpret text with remarkable fluency and contextual understanding has reshaped how machines engage with human language. At the core of this transformation lies the transformer architecture, a groundbreaking framework for sequence modeling that has significantly advanced NLP research. By leveraging this architecture, LLMs can process and analyze complex textual patterns with an unprecedented level of precision [63]. This part explores the fundamental principles of LLMs and their role in strengthening blockchain security.

Prompting Paradigms for Vulnerability Detection

LLMs can be adapted to detect vulnerabilities in smart contracts through various prompting strategies. These strategies typically fall into three main categories: binary, multi-class, and open-ended prompting. In binary prompting, the model produces a yes/no response indicating whether a vulnerability exists. Multi-class prompting expands this by allowing the model to classify the input into one of several predefined vulnerability types. Open-ended prompting, on the other hand, enables the model to generate descriptive outputs or identify novel or complex vulnerabilities beyond a fixed label set [113]. Each approach serves different stages of smart contract analysis, balancing interpretability and flexibility.

Language Models for Secure Smart Contract Analysis

Transformer-based language models, particularly those developed under the Generative Pre-trained Transformer (GPT) framework, have revolutionized natural language processing by capturing deep semantic and syntactic relationships through attention mechanisms. These models are pre-trained on massive corpora to learn general language patterns and are then fine-tuned for domain-specific applications. Among the most prominent are OpenAI's GPT series and Meta's LLaMA models [118, 119], with GPT-4 and its optimized variant GPT-4o standing out for their superior performance and widespread adoption.

In this research, GPT-4o is used as a foundation for real-time vulnerability detection within a VSCode-based development environment. This model integrates seamlessly with the developer workflow, providing immediate feedback and remediation suggestions. Its deployment in our plugin demonstrates the viability of incorporating powerful LLMs into everyday coding

tools, thereby enhancing the security posture of smart contract developers. Beyond API-driven detection, our research also explores model independence through the fine-tuning of LLaMA-3.1-8B on an enriched dataset of labeled vulnerabilities. This alternative offers a scalable and locally deployable solution, ensuring broader accessibility and reducing reliance on closed-source infrastructure. Collectively, our work underscores the transformative potential of LLMs like GPT-4o and LLaMA in blockchain security. By combining advanced reasoning mechanisms with domain-specific adaptation, we establish a framework capable of both high accuracy and practical integration for smart contract vulnerability detection.

Parameter-Efficient Fine-Tuning (PEFT)

Fine-tuning LLMs poses significant computational challenges due to their immense scale, often ranging from 7 to 65 billion parameters. Fully fine-tuning such models requires substantial memory and computing resources. This complexity arises from the need to load the entire model in full precision and update all parameters across large-scale datasets.

To address these limitations, PEFT techniques have been developed as a viable alternative. These methods aim to retain the benefits of task-specific adaptation. At the same time, they dramatically reduce the number of trainable parameters involved in the optimization process. Rather than updating the full model, these techniques selectively introduce a small set of trainable components—such as adapters, bias terms, or low-rank matrices—into specific layers of the network. By keeping the majority of the model weights frozen, PEFT enables fine-tuning with a fraction of the computational and memory overhead. This approach not only improves efficiency but also promotes better generalization and stability, especially when training on limited data or with constrained resources.

Low-Rank Adaptation (LoRA) [125] represents a prominent and widely adopted method within the PEFT paradigm. LoRA enhances the efficiency of model adaptation by injecting trainable low-rank matrices into the attention mechanism of transformer-based architectures. Specifically, LoRA decomposes the weight updates into two low-rank matrices that are inserted in parallel with the frozen original weights in attention layers. During fine-tuning, only these low-rank matrices are updated, while the pretrained parameters remain unchanged.

This design significantly reduces the number of trainable parameters and the associated memory footprint, enabling scalable fine-tuning even on commodity hardware. Moreover, LoRA preserves the expressive power of the model by leveraging the representational capacity of the frozen weights while introducing just enough flexibility to learn task-specific patterns. LoRA belongs to the broader family of adapter-based techniques, which similarly insert lightweight modules into neural network architectures to support efficient transfer learning.

Its simplicity, effectiveness, and ease of integration have made LoRA a foundational approach for practical and efficient fine-tuning of LLMs across a wide range of applications.

9.3 Related Work

In this section, we investigate existing works focusing on smart contract vulnerability detection using LLMs. In our previous work [4], we introduced an advanced framework for detecting smart contract vulnerabilities using GPT-based language models. Our methodology included extensive dataset refinement, such as removing biased comments, anonymizing function and variable names, and restructuring data for realistic evaluation. We employed sophisticated prompt engineering, combining expert-driven patterns with chain-of-thought (CoT) reasoning and few-shot examples. The resulting outputs provided comprehensive details, such as identification of vulnerable code lines, exploit scenarios, and recommended fixes. The evaluation was conducted using both quantitative metrics and expert reviews, ensuring a high standard of accuracy and security.

Despite these strengths, several limitations persist. First, the reliance on the GPT-API results in significant operational costs. Additionally, while the underlying dataset is meticulously curated and normalized, it remains relatively modest in size and is constructed through the injection of synthetic bugs into real-world smart contracts. This process, although it enhances data quality, may limit the breadth and real-world representativeness of the evaluation, potentially affecting the generalizability of the findings. Another limitation is that the approach focuses on static vulnerability detection and does not operate in real time.

To overcome the limitations of dataset scope, cost, API availability, and real-time applicability, this work presents a comprehensive dataset, a real-time detection tool, and the integration of fine-tuned large language models. The proposed approach enhances detection performance by fine-tuning GPT and LLaMA models, offering a more robust, accessible, and cost-effective solution for smart contract vulnerability identification.

Ding et al. [167] propose SmartGuard, an LLM-based vulnerability detection framework utilizing the SolidiFi dataset and CoT prompting. However, SmartGuard applies LLMs directly to the unrefined dataset and relies on CodeBERT-based semantic retrieval for prompting, without explicit expert guidance or rigorous dataset preprocessing. Moreover, its evaluation methodology lacks expert validation and does not address dataset leakage or cost-efficiency considerations.

Ma et al. [129] propose iAudit, a smart contract auditing framework that leverages a two-stage fine-tuning process integrated with LLM-based agents. While techniques such as majority

voting and agent-based feedback loops are employed to mitigate hallucinations, the system remains vulnerable to generating inaccurate results.

Moreover, the iterative reasoning process between the Ranker and Critic is constrained by a fixed number of steps, potentially limiting the depth and refinement of explanations. A notable limitation of this approach is its low consistency, which poses a significant challenge to its reliability. Another major shortcoming lies in their dataset, which is both limited in size and scope. It comprises only 263 contracts, with a median length of approximately 35 lines—significantly shorter than typical real-world smart contracts—raising concerns about the generalizability and robustness of their evaluation.

Wei et al. [6] propose a framework named *LLM-SmartAudit*, which employs a multi-agent conversational approach, using specialized agents to collaboratively detect and analyze vulnerabilities in smart contracts. *LLM-SmartAudit* lacks a well-defined benchmark dataset like SolidiFi, making direct comparisons with other methods difficult. Secondly, it primarily relies on LLM knowledge, which can lead to hallucinations, especially when encountering novel vulnerabilities. Moreover, it does not explicitly pinpoint the exact lines of vulnerable code, making it less precise than approaches that provide detailed localization. Additionally, the system does not suggest specific code fixes, limiting its usefulness for developers who need remediation guidance.

In contrast, our previous work [4] leverages the SolidiFi benchmark dataset and performs preprocessing to remove biases, ensuring a more reliable evaluation. It also achieves higher recall rates. Furthermore, unlike *LLM-SmartAudit*, this method identifies the exact lines of vulnerable code and proposes fixes, making it more actionable for security professionals. Using a chain-of-thought (CoT) prompting strategy enhances the reasoning process, improving the detection accuracy. However, it does not employ a multi-agent system, a key strength of *LLM-SmartAudit*.

Hu et al. [113] present a two-stage framework called GPTLens that leverages large language models for smart contract analysis, where the model functions both as an auditor and as a critic to enhance detection quality. Although the method helps reduce false positives, the first stage tends to produce an excessive number of candidate vulnerabilities, which may burden the analysis process or lead to inefficiencies in downstream validation.

Yu et al. [169] introduce Smart-LLaMA, which Focuses on domain adaptability, proposing a two-stage post-training process for LLMs tailored to smart contract vulnerability detection. By constructing a comprehensive dataset with detailed explanations and precise vulnerability locations, and through smart contract-specific continual pre-training followed by explanation-guided fine-tuning, Smart-LLaMA enhances both detection performance and the quality of

explanations provided.

Zaazaa et al. [166] also leverage LLMs, specifically ChatGPT with in-context training to enhance vulnerability detection by streamlining the integration of new detection rules. This framework has demonstrated an exact match accuracy of 91.1% with sufficient data, showcasing its potential in improving blockchain security. Liu et al. [164] propose PropertyGPT, an approach that employs large language models to detect vulnerabilities in smart contracts, evaluated on a small subset of 13 contracts from the *Smart-Bugs Curated* dataset [128]. Their method outperforms GPTScan [165] by successfully identifying vulnerabilities in 9 contracts, whereas GPTScan detects issues in only 5. Despite this improvement, PropertyGPT is constrained to binary classification and addresses a limited set of vulnerability types. In contrast, GPTScan [165] integrates GPT-3.5-turbo with static analysis to locate potential vulnerabilities. Yet, its performance may be hindered by the accuracy limitations inherent in static analysis, particularly when applied to more complex or extensive contracts.

Ince et al. [168] explore vulnerability detection in smart contracts by fine-tuning two large-scale open-source models—Meta's Code Llama and Instruct variants—each with 34 billion parameters. While the models were trained to identify security flaws, the reliance on underlying tools with known low detection efficacy led to inconsistent results across different vulnerability categories.

Ghaleb et al. [170] present SolidiFI, a benchmarking suite designed to assess smart contract vulnerability detection tools by injecting synthetic bugs into authentic contracts retrieved from *Etherscan*[1]. Building on this, Du et al. [5] evaluate the applicability of GPT-4 on the SolidiFI dataset and conclude that the model struggles with effective vulnerability detection. However, their limited use of prompt engineering appears to undermine the model's capabilities, leading to performance that, in some cases, falls below that of random classifiers. Unlike these binary classification approaches, our method incorporates structured reasoning to enhance both detection accuracy and explanation quality.

These frameworks represent significant advancements in the application of LLMs to smart contract vulnerability detection, each contributing unique methodologies to enhance security in blockchain technologies. Table 9.1 provides a comparative summary of the existing approaches.

[1] `https://etherscan.io`

Table 9.1 Comparing recent LLM-based smart contract vulnerability detection frameworks.

Reference	LLM Model	Framework	Technique	Limitations
Ghaleb et al. (2020) [170]	N/A	SolidiFI	Benchmarking and dataset generation	Not a detection system; focuses on benchmarking
Hu et al. (2023) [113]	GPT-based (API)	GPTLens (Auditor + Critic)	Two-stage LLM auditing; reduces false positives	High volume of candidates in first stage; validation bottleneck
Liu et al. (2024) [164]	GPT-based (API)	PropertyGPT	LLM prompt-based binary detection	Small sample size; only binary detection; limited vuln.* types
Sun et al. (2024) [165]	GPT-3.5-turbo	GPTScan	LLM + static analysis for detection	Limited by static analysis precision; struggles on large/complex contracts
Zaazaa et al. (2024) [166]	ChatGPT	SmartLLMSentry	Rule-based detection with in-context LLM	Dependent on sufficient data; not evaluated for localization
Ince et al. (2024) [168]	Code Llama (34B)	Fine-tuned LLM	Fine-tuned LLMs for vulnerability detection	High hardware cost (A100 GPUs); inconsistent detection
Ma et al. (2024) [129]	GPT-3.5	iAudit	Two-stage, majority voting, agent feedback to mitigate hallucination	Small, short contracts; low output consistency; generalizability concerns
Wei et al. (2024) [6]	GPT-based (API)	LLM-SmartAudit	Multi-agent conversational detection	No established benchmark; lacks localization and fix suggestions
Yu et al. (2024) [169]	LLaMA-3.1 (8B)	Smart-LLaMA	Domain-adapted pretraining + explanation-guided fine-tuning	Generate repetitive explanations after FT*
Du et al. (2024) [5]	GPT-4 (API)	Prompt-based GPT-4	Prompt-based GPT-4 on benchmark	Weak prompt engineering; underperforms random classification
Erfan et al. (2024) [4]	GPT-based (API)	Advanced SCVD* with dataset refinement	Expert-driven prompt engineering, CoT, few-shot, dataset anonymization	API cost, static detection, moderate dataset size, dependent on APIs
Ding et al. (2025) [167]	LLM + CodeBERT	SmartGuard	CoT*, CodeBERT semantic retrieval	Unrefined dataset; lacks expert validation; no cost analysis; no explicit expert guidance; less robust prompt design

Abbreviations: FT: fine-tuning, vuln.: vulnerability, CoT: chain of thought, SCVD: smart contract vulnerability detection

9.4 The Proposed Approach

Erfan et al. [4] employed GPT-4o APIs to detect vulnerabilities in smart contracts, utilizing CoT reasoning in conjunction with the SolidiFi dataset. While this dataset offered precise annotations, it was synthetically generated and lacked the variability and complexity of real-

world smart contracts, thereby limiting the model's generalizability to practical scenarios. Moreover, the approach relied exclusively on prompt engineering without direct fine-tuning, which constrained the model's capacity to capture deeper vulnerability patterns.

In the present study, we address these limitations by constructing a comprehensive and cleaned labeled dataset, aggregating several state-of-the-art sources of smart contract vulnerabilities. Using this dataset, we fine-tune an open-source large language model, *LLaMA-3.1-8B*, employing LoRA to enable parameter-efficient training. This fine-tuning enhances the model's ability to detect vulnerabilities with greater precision and contextual understanding. Additionally, we extend our previous API-based approach by implementing it as a VSCode plugin, enabling real-time security analysis within the developer environment. The key phases of this work are detailed below.

9.4.1 Dataset Development and Preparation

Constructing a comprehensive and reliable dataset remains one of the primary challenges in the domain of smart contract security. This difficulty stems from the limited availability of high-quality, publicly accessible datasets. Many existing resources are either commercial, incomplete, or lack reliability due to their dependence on automated analysis tools with high false positive rates [3, 127]. As highlighted in earlier sections, these tools often fail to provide accurate ground truth labels, and most existing datasets [5, 62, 128, 129] do not include essential elements such as the specific lines of code associated with vulnerabilities or the corresponding vulnerable code segments.

Furthermore, the reliability of some prior studies is undermined by the use of limited or incomplete datasets. A large-scale, diverse, and well-structured dataset is essential to support effective fine-tuning and evaluation of LLMs. To address these limitations and enable reliable training and evaluation of large language models, we developed a comprehensive dataset through a systematic multi-stage pipeline. Our dataset development and preparation pipeline encompasses the following core tasks:

Data Gathering To fine-tune an open-source LLM for smart contract vulnerability detection, we curated a rich dataset comprising labeled smart contracts annotated with detailed explanations. These include the rationale behind each vulnerability, the corresponding exploitation method, and the recommended remediation. The dataset focuses on three critical vulnerabilities: Reentrancy (RE), Integer Overflow/Underflow (IoU), and Timestamp Dependency (TD). Each sample contains the original vulnerable code, its fixed counterpart, and a comprehensive explanation of the associated risk. Figure 9.1 provides an overview of

the data sources used. Despite relying on state-of-the-art resources widely adopted in academic and developer communities, extensive preprocessing and validation were required due to inconsistencies and noise.

Our final dataset comprises 12 structured JSON files, aggregated from the following sources:

- Our previous work [4] based on the SolidiFi dataset [5];
- FTSmartAudit dataset [6];
- Timestamp dependency data from [2];
- Reentrancy dataset from [7];
- Multi-vulnerability dataset including RE, TD, and IoU from [1];
- Reentrancy dataset from SmartBugs [3];
- Line-level annotations obtained from the dataset provided by [169] were used to enrich datasets such as [1–3, 7].

Data Cleaning To ensure dataset consistency and improve generalizability, we applied several preprocessing steps, including the removal of redundant metadata and standardization of naming conventions. Comments and identifiers that could reveal vulnerability types (e.g., function names such as `vul`, `RE`, `IoU`, or `TD`) were removed to prevent LLMs from exploiting these clues. Moreover, imprecise labels were corrected, and a new field, `vulnerableCode`, was introduced to explicitly capture the code lines associated with each vulnerability.

Compound Auditing This step involves a two-phase process to ensure the quality and completeness of vulnerability annotations. In the first phase, we applied our previous methodology [4], which leverages GPT-based CoT prompting with in-context examples and targeted instructions. Through this approach, the GPT model generated detailed textual explanations addressing three core aspects for each code sample: (1) the rationale for labeling a specific line as vulnerable, (2) the potential exploitation strategies an attacker might use, and (3) actionable recommendations or fixed code snippets for remediation.

In the second phase, all generated outputs underwent a thorough review by a domain expert. The expert critically examined the model-generated explanations, identified any inaccuracies or ambiguities, and ensured that each annotation provided clear, accurate, and contextually appropriate guidance. This expert-in-the-loop strategy enabled us to validate and, where

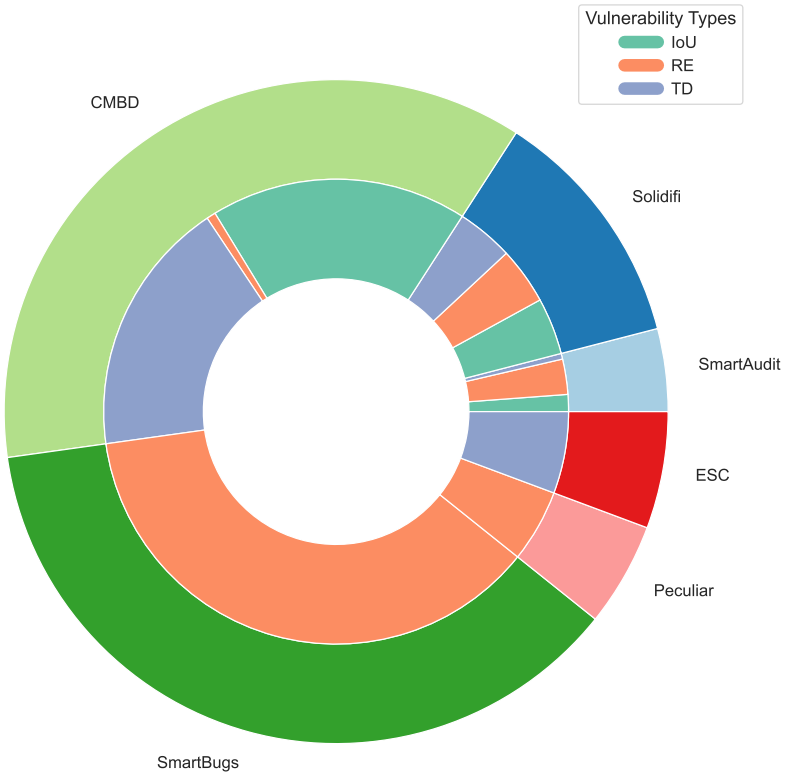


Figure 9.1 Dataset resources containing CMBD: cross modularity bug detection [1], ESC: Ethereum Smart Contracts [2], SmartBugs [3], SolidiFi [4, 5], SmartAudit [6], Peculiar [7].

necessary, refine the dataset, particularly for instances where original annotations were incomplete (such as missing explanations or repair suggestions).

By combining automated GPT-driven reasoning with rigorous human expert review, our compound auditing process achieved consistent and comprehensive explanatory coverage across the dataset. This hybrid approach not only augmented the quality of vulnerability type descriptions, security risk assessments, and remediation strategies but also improved the reliability and usability of the dataset for downstream fine-tuning and evaluation tasks.

Format Standardization To facilitate model training and evaluation, we unified the dataset schema across all sources. Each entry includes the following standardized fields: `vulnerableLines`, `vulnerableCode`, `fixedCode`, `vulnerabilityReason`, and `potentialSecurityRisk`. This standardization enables efficient dataset loading and downstream model compatibility.

Data Balancing We augmented the dataset with verified safe smart contracts to prevent class imbalance and support generalization. These examples were manually selected from real-world repositories and include secure implementations of patterns corresponding to the identified vulnerabilities.

Instruction Creation For each vulnerability type, we designed a set of instruction prompts to support instruction-tuning of LLMs. These prompts are accompanied by representative examples of both vulnerable and secure code, facilitating task-specific adaptation of the model.

Storing Dataset Variations We created multiple formats to support diverse experimental workflows: (1) a fine-tuning-ready version with `prompt`, `input`, and `output` fields; (2) an Alpaca-style format tailored for LLaMA models; (3) a comprehensive version including all label fields; and (4) a function-level version where contracts are segmented into their individual functions for fine-grained analysis.

Our datasets include both vulnerable and safe code samples. The complete dataset repository, including training, validation, and test splits, is publicly available [2].

9.4.2 Fine-Tuning the Smart Contract Vulnerability Detection Model

While our initial approach relied on proprietary models like GPT-4o to detect, reason about, and fix vulnerabilities in smart contracts, it introduced important challenges. These models are expensive to run, which limits scalability, and they require sending sensitive contract data to external APIs, raising concerns about data privacy and confidentiality—especially when handling unreleased or security-critical code. To address these issues, we propose shifting to open-source solutions by fine-tuning a large language model specifically for smart contract security. This approach significantly reduces computational costs, allows for local deployment to ensure better data protection, and provides greater control for customizing the model to fit domain-specific needs without depending on third-party services.

Our fine-tuning methodology is built upon *LLaMA-3.1-8B-Instruct* [233], an instruction-tuned language model that we further adapt using supervised fine-tuning (SFT) with LoRA-based parameter-efficient training. This process allows for effective specialization on domain-specific tasks while minimizing resource consumption. The goal of this fine-tuning process is to enable the model to identify security vulnerabilities in smart contract code and generate structured outputs, including the vulnerable code lines, an explanation of the issue, the

[2] `https://github.com/erfan38/FT_LLM_SCVD.git`

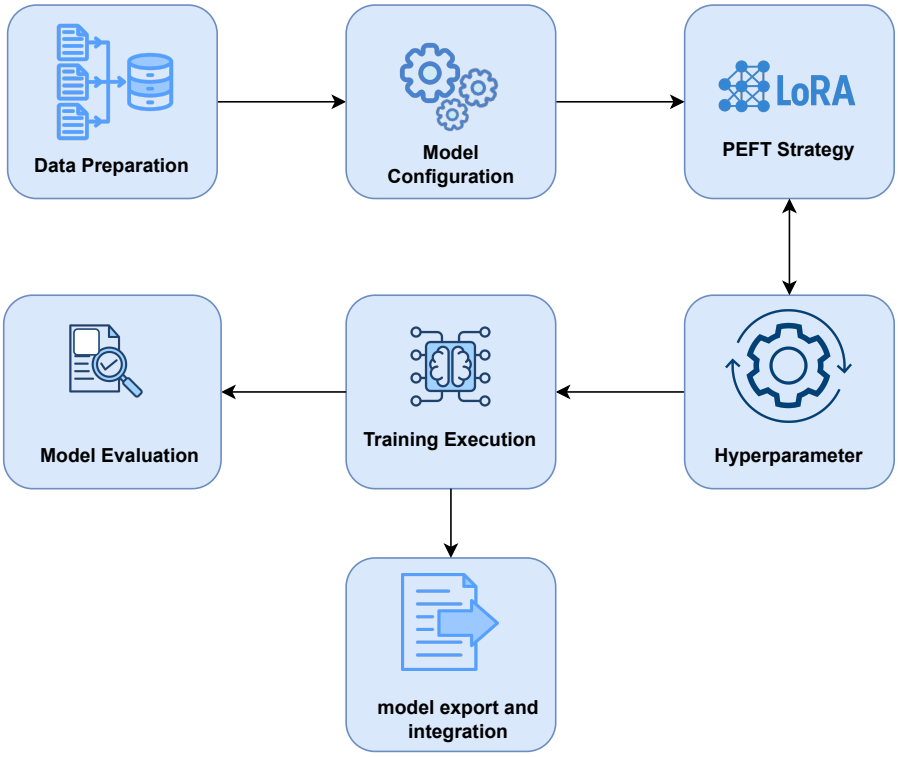


Figure 9.2 High-level overview of our research methodology.

associated security risk, and the recommended fix. As illustrated in Figure 9.2, the fine-tuning process consists of seven steps, including data preparation, model configuration, parameter-efficient fine-tuning, hyperparameter determination, training execution, model evaluation, and model export and integration, which are explained as follows.

Dataset preparation

We use our, multi-source dataset containing expert-crafted instructions for various smart contract vulnerabilities. Each instruction includes a detailed vulnerability description, detection guidance, and examples of both secure and insecure code, following the patterns defined in [4]. The combined training corpus includes:

- Instruction-only records for generic reasoning.
- Function-level samples with ground truth annotations.

Each data sample was stored in JSONL format, containing fields such as `vulnerableLines`,

`vulnerableCode`, `vulnerabilityReason`, `fixedCode`, and `potentialSecurityRisk`. These structured fields ensured that the model learned not only to detect vulnerabilities but also to reason about them and suggest remediation actions.

Model configuration

We employed LLaMA Factory [234], an open-source framework for instruction tuning, to manage the fine-tuning process. The selected base model was *LLaMA-3.1-8B-Instruct*, a pre-trained and instruction-aligned model capable of handling long-context reasoning. To accommodate the complexity of smart contract functions, many of which contain nested logic, modifiers, and lengthy blocks of code, the input sequence length was configured with a cutoff of 4096 tokens. This ensured that the model had sufficient context to capture both the vulnerable code and its surrounding logic, which is often critical for accurate vulnerability detection and explanation.

Furthermore, training was conducted using *bfloat16 (bf16)* precision, a low-precision floating point format that maintains numerical stability while significantly reducing memory consumption and improving computational efficiency. This choice enabled more efficient training of the large model, particularly when used on hardware supporting native bf16 operations.

Parameter efficient fine tuning strategy

We employed LoRA as a PEFT strategy for efficient training on resource-constrained hardware. LoRA introduces trainable low-rank matrices into the transformer layers while keeping most of the base model parameters frozen. In this configuration, we set the rank of the adaptation matrices to 8. We used a scaling factor (alpha) of 16, which controls the magnitude of the updates introduced by the LoRA layers. Additionally, LoRA adaptation was applied across all transformer layers, enabling uniform adaptation throughout the model architecture.

We also unfroze two transformer layers, enabling a small portion of the original model parameters to be updated during training to allow limited and targeted base-model adaptation. Despite these modifications, the number of trainable parameters remained low—approximately 20.97 million, which corresponds to only 0.26% of the total model size of 8.05 billion parameters. This setup allowed for efficient fine-tuning with minimal impact on model quality and without requiring large-scale computational resources.

Hyperparameter determination

We used the AdamW optimizer, a refined version of the popular Adam optimization method, to train the model effectively and ensure it generalizes well to new smart contract vulnerabilities. AdamW is specifically designed to improve training stability and reduce overfitting, which can occur when a model learns patterns too specific to the training data. Unlike earlier versions of Adam, AdamW more effectively manages how much the model relies on its previous weights during training. This makes it especially suitable for fine-tuning large language models, where maintaining a balance between learning new task-specific knowledge and preserving general capabilities is crucial [235].

To further support smooth training, we applied a cosine learning rate schedule, gradually reducing the learning rate as training progresses. This allows the model to learn quickly at the start and then fine-tune its understanding more precisely over time. We used a small initial learning rate ($5e-5$) to ensure careful adjustments to the model, particularly since most of the model parameters were frozen and only a small set of LoRA adapters were being updated.

We increase the maximum input length to 4096 tokens to ensure the model can process complete smart contract functions, many of which span multiple lines and contain nested structures. This longer context window allows the model to fully capture both the vulnerable code and the surrounding logic, which is often critical for accurate vulnerability detection and reasoning. Since longer inputs require more memory, we used a small batch size of 2 to remain within available hardware limits. To compensate for the small batch size and maintain training effectiveness, we applied gradient accumulation over 8 steps. This technique simulates a larger batch size by accumulating gradients across multiple forward passes before performing an update. This configuration balanced convergence, stability, and memory efficiency, which is particularly important given the model's size and token context window.

Our fine-tuned model is designed for practical deployment in order to facilitate integration into real-world development pipelines and tools. The low number of trainable parameters, combined with BF16 precision and LoRA adapters, significantly reduces memory usage and latency. This makes the model suitable for deployment on edge or decentralized systems, supporting real-time auditing of smart contracts without reliance on APIs.

Training execution

The training prompts are formatted according to the *LLaMA 3* instruction prompt structure, which aligns with single-turn tasks involving an inst. We apply tokenization using the tok-

enizer associated with the LLaMA-3.1-8B model to convert the raw text data into a format suitable for processing by the transformer architecture. Then, ruction, an optional input, and a target output. The system periodically evaluated the model's performance on the validation dataset during training, allowing us to monitor trends in loss and generalization in real time.

Model evaluation

The model is evaluated using two categories of techniques. The first involves standard classification metrics, including precision, recall, and F1-score. These metrics are applied to assess the model's ability to localize vulnerabilities within the code correctly and to determine whether a given function should be flagged as vulnerable or safe. Since the LLM is designed to specify the exact lines of code affected by a vulnerability, a prediction is considered correct if the identified lines fall within or overlap with the ground-truth annotated vulnerable code region. It is important to note that the model typically returns a segment of code starting from the vulnerability trigger line and extending through the affected function body.

The second category of evaluation focuses on assessing the quality of the generated text. We employ several metrics based on cosine similarity, including consistency, code similarity, and risk assessment alignment, to evaluate different aspects of the model's outputs [236, 237]. Consistency measures the semantic alignment between the model-generated explanation (vulnerability reason) and the human-annotated reference. By computing the cosine similarity between their sentence embeddings, we assess the degree to which the model's reasoning is factually accurate and consistent with expert annotations.

Code similarity assesses whether the model-generated fixed code closely aligns with the ground truth fix provided in the labeled data. Specifically, we calculate the cosine similarity between the embeddings of the predicted fixed code and the reference fixed code. A higher similarity score indicates that the model's fix is contextually relevant and consistent with the intended remediation, whereas a lower score may suggest the fix is either generic or deviates from the reference solution.

To evaluate the semantic alignment between the model's predicted potential security risk and the ground truth label, we compute the cosine similarity between their respective embeddings. This metric quantifies the degree of semantic coherence between the predicted risk description and the reference, indicating how well the model's output aligns with the expected security implications. For embedding generation, we utilize the `all-MiniLM-L6-v2` model from the Sentence Transformers library [237]. This model efficiently maps textual inputs to a 384-dimensional dense vector space, facilitating effective semantic comparisons. All evaluation

results are reported in Section 9.5.

Model export and integration

At the end of training, all necessary components required for downstream integration and inference were exported. This included the fine-tuned LoRA adapter weights, which captured the task-specific modifications introduced during fine-tuning while preserving the original parameters of the base model. The tokenizer and configuration files were also saved to ensure consistency in how input data is processed during inference. With these components in place, the model is capable of analyzing smart contract code and generating structured vulnerability assessments. The fine-tuned model's output includes the specific lines of code identified as vulnerable, the reasoning behind their classification, the associated security risks, and a proposed fix. This structured output can be readily integrated into automated security analysis pipelines and supports practical applications in smart contract auditing.

9.4.3 VSCode Plugin for Smart Contract Security Analysis

We designed and implemented a Visual Studio Code (VSCode) plugin that integrates real-time LLM-assisted smart contract vulnerability detection directly into the development environment. This plugin enhances developer productivity and improves the security of smart contract development workflows. The plugin leverages the capabilities of an LLM, specifically the GPT-4o and GPT-4o-mini APIs, to semantically analyze solidity smart contract code and identify common vulnerabilities, along with the vulnerability reasons, the potential risk, and provide the fixed code for each vulnerable code. In contrast to static analyzers that depend on hardcoded rules or symbolic execution, our plugin formulates rich contextual prompts, including in-context examples and detailed vulnerability instructions, to guide the LLM in performing deep reasoning over the provided source code.

Upon initialization, the plugin prompts the user to input a valid OpenAI API key, which is securely stored locally for subsequent interactions. Users can select from a predefined list of vulnerability types, namely Integer Overflow/Underflow, Timestamp Dependency, and Reentrancy, each corresponding to a specific risk class in Ethereum smart contract security. When a solidity smart contract is opened or modified in the editor, the plugin captures the current code, constructs a specialized prompt using both instruction-based reasoning and few-shot training data, and sends the request to the LLM via the OpenAI API.

Internally, the plugin architecture relies on a `Generator` class, which dynamically configures the system message, user instruction, and response format based on the “detector" roles. The

detector identifies vulnerable lines of code, explains the nature of the vulnerability, outlines the potential security risk, and returns a corresponding fix. The output is formatted as a structured RFC8259-compliant JSON object to facilitate further processing.

The core `Generator` logic uses an LLM prompt completion strategy, where user instructions are tailored per role and vulnerability. Prompt responses are retrieved using the GPT-4o API and subsequently parsed for actionable insight.

The plugin provides two integrated user interfaces to enhance usability. Within the code editor, detected vulnerabilities are highlighted inline, accompanied by hoverable annotations describing each issue. Additionally, a dedicated side panel aggregates all detection results into a structured list, with interactive elements to apply or reject suggested fixes. Upon confirmation, secure versions of the vulnerable code are programmatically injected into the editor, streamlining the remediation process.

The plugin also includes auxiliary features that improve transparency and robustness. These include a memory usage logger that tracks system resource consumption before and after the detection process, a batch testing mode for evaluating multiple contracts in a single run, and runtime performance logging to capture latency and UI rendering times. Internally, the detection pipeline utilizes a modular prompt generation system supported by a Generator class, which dynamically inserts relevant examples.

This plugin supports multiple output formats, including full JSON reports suitable for downstream analysis or benchmarking. The modular design enables easy extension to additional vulnerability types or integration with other LLM providers.

As shown in Figure 9.3, the VSCode plugin provides real-time semantic analysis of Solidity smart contracts by leveraging an LLM. When a developer opens or edits a contract file, the plugin identifies security vulnerabilities and highlights the affected lines directly within the editor. In the presented example, three critical vulnerabilities are detected:

- *Integer Overflow/Underflow:* Line 7 contains an unchecked arithmetic operation on `lockTime[msg.sender]`, which may result in an overflow and allow premature withdrawals.
- *Reentrancy:* Line 12 uses the `transfer` function to send Ether before any state change, making the contract susceptible to reentrancy attacks.
- *Time Dependency:* Line 11 relies on `block.timestamp`, which can be manipulated by miners to bypass time constraints and release funds earlier.

The right-hand panel provides structured vulnerability reports, each with a brief explanation of the issue and its potential consequences. Developers can interactively respond to each vulnerability through the interface by choosing either to accept the fix generated by the model or to decline it. This visualization demonstrates how our plugin integrates LLM-powered vulnerability detection into practical smart contract development workflows, improving both code security and developer experience.

Overall, this plugin serves as a lightweight and effective tool for integrating AI-powered security auditing into real-world development workflows. By providing immediate, actionable feedback and facilitating on-the-fly code correction, the system supports a proactive approach to vulnerability mitigation. It bridges the gap between secure programming practices and practical developer tooling.

In summary, the developed plugin builds upon previous work by advancing from API-dependent vulnerability detection to a fully fine-tuned, open-source solution tailored for smart contract security analysis. The goal is to advance the security of blockchain applications by bridging the gap between API-based solutions and practical, open-source, AI-driven auditing tools. We introduced a real-time auditing tool via a VSCode plugin, curated a comprehensive and structured dataset enriched with explanatory reasoning, and fine-tuned the *LLaMA-3.1-8B-Instruct* model using parameter-efficient training strategies such as LoRA. The fine-tuning pipeline was optimized for both performance and deployability, enabling the resulting model to accurately detect vulnerabilities, explain their causes, and recommend remediation in a structured and interpretable format. With the trained model fully exported and ready for integration, we now evaluate its effectiveness through a series of experiments and quantitative analyses.

9.5 Experimental Results

In this section, we present the evaluation of our proposed approach alongside our implemented VSCode plugin. First, we conduct a comprehensive analysis of the performance of our fine-tuned model using multiple evaluation metrics to assess its effectiveness in detecting smart contract vulnerabilities. In the range of LLMs with small parameters (around 7-8 billion parameters), we choose *Llama 3.1* [233] with eight billion parameters and GPT-4o-mini [238] with the same range of parameters. Then, we fine-tuned both models and compared them with their original models. We report classification accuracy, precision, recall, F1-score, and semantic alignment metrics (explained in Section 9.4.2), including code similarity, consistency, and risk assessment alignment.

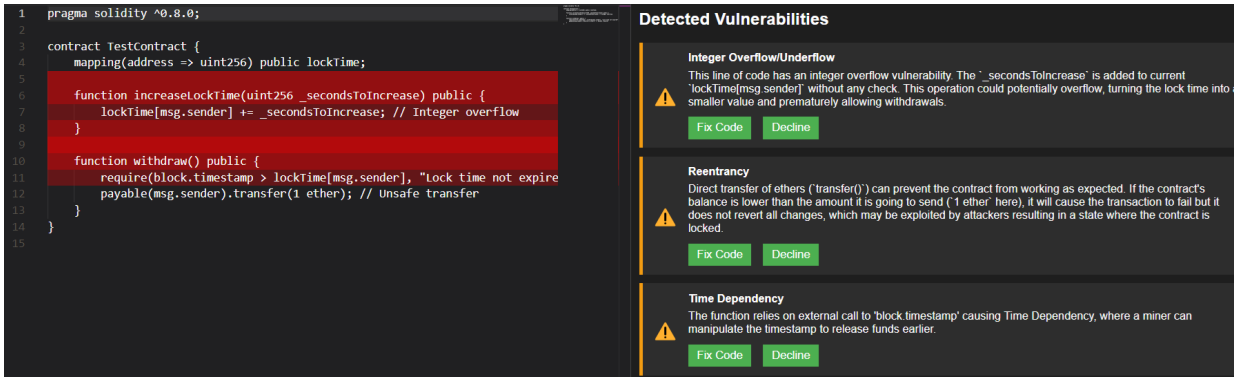


Figure 9.3 Overview of the VSCode plugin detecting multiple vulnerabilities in a Solidity smart contract. Detected issues are highlighted directly in the editor, with explanations, risk assessments, and fix suggestions presented in the side panel.

Our experiment leverages our dataset of 1266 vulnerable smart contracts, partitioned into 949 for training, 191 for testing, and 126 for validation. For this study, we use a function-based version of the dataset (as described in Section 9.4.1), comprising 8462 training, 1628 test, and 1200 validation samples including vulnerable and safe functions. As previously stated, we focus on the top three smart contract vulnerabilities: IoU, TD, and RE.

We computed the confusion matrix based on the test set comprising 1,628 samples in order to evaluate our vulnerability detection system. The results presented in Figure 9.4 illustrate the correspondence between predicted and actual labels for the line-level vulnerability detection task, where each line of code is classified as either vulnerable or non-vulnerable. This matrix displays the relationship between predicted and actual classes for discovering vulnerable code lines. The confusion matrix yields True Negatives (TN) of 762, representing non-vulnerable instances correctly identified as non-vulnerable; False Positives (FP) of 52, indicating non-vulnerable instances incorrectly classified as vulnerable; False Negatives (FN) of 159, which are vulnerable instances incorrectly classified as non-vulnerable; and True Positives (TP) of 655, referring to vulnerable instances correctly identified as vulnerable.

As summarized in Table 9.2, our model demonstrates strong performance in detecting vulnerable lines, particularly in minimizing false positives, as indicated by the high precision. The recall is also reasonably high, confirming the model's ability to detect a significant portion of actual vulnerabilities. The balanced F1-score further reinforces the model's effectiveness in handling both vulnerable and non-vulnerable classes.

The semantic similarity between the text-based outputs generated by our model and the corresponding reference labels was systematically evaluated. Specifically, embedding-based similarity scores were calculated for a total of 1,628 test samples. As presented in Table 9.3,

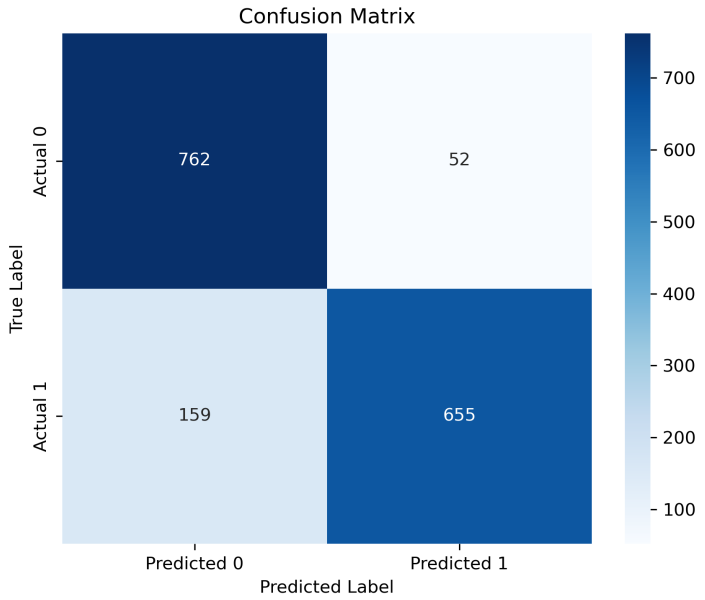


Figure 9.4 Confusion matrix of the vulnerability detection model on the test dataset. The matrix illustrates the distribution of predicted versus actual classes.

the alignment between the predicted and reference texts was assessed across four critical components: *vulnerability reason*, *potential security risk*, *fixed code*, and *vulnerable code*.

To compare the models effectively, we evaluated them using a shared subset of the test dataset, ensuring a fair and controlled basis for comparison. As shown in Table 9.4, our evaluation includes both base and fine-tuned versions of the Llama-3.1-8B and GPT-4o-mini models, as well as the newly released GPT-4.1-mini.

Across all standard metrics—accuracy, precision, recall, and F1-score—our fine-tuned Llama3.1 model demonstrates substantial improvements over both its base counterpart and the GPT baselines. Notably, the accuracy of our fine-tuned Llama3.1 reaches 90%, a dramatic increase from the 50% achieved by the original, non-fine-tuned Llama3.1 model. This improvement exceeds the performance of the much larger and more recently released GPT-

Table 9.2 Evaluation metrics of the model on 1628 test samples.

Metric	**Score (%)**
Accuracy	84.95
Precision	92.69
Recall	80.44
F1-Score	86.13

Table 9.3 Evaluation of generated text alignment with reference labels across 1,628 test samples

Metric	**Score (%)**
Vulnerability Reason	78.12
Potential Security Risk	73.69
Fixed Code	89.76
Vulnerable Code	95.14

4.1-mini (76.00%).

Fine-tuning the open-source Llama3.1 model yields a markedly greater performance gain than what we observe for GPT-4o-mini. While the fine-tuned GPT-4o-mini achieves only a modest improvement over its base version (60.00% vs. 56.00% accuracy), fine-tuning Llama3.1 nearly doubles the accuracy of its base model. This trend is consistently reflected across all metrics, including precision (93.55% vs. 60.60%), recall (90.63% vs. 63.16%), and F1-score (92.06% vs. 58.54%). Beyond classification metrics, our fine-tuned Llama3.1 also attains the highest consistency and similarity scores in both vulnerability reasoning and fixed code generation. These findings are visually summarized in Figure 9.5, which highlights the relative improvements from fine-tuning. While fine-tuning yields only marginal gains for GPT-4o-mini, the improvement for Llama3.1 is both significant and pronounced.

Furthermore, the rationale for selecting Llama as the core model for this work is rooted in its open-source nature. Llama models allow for full local fine-tuning and transparent control over the training pipeline, enabling secure handling of sensitive datasets—an essential requirement for security-critical domains such as smart contract vulnerability detection. In contrast, LLMs such as GPT-4o-mini and GPT-4.1-mini do not offer the same degree of transparency or local fine-tuning capability, potentially raising privacy and reproducibility concerns.

In summary, the results clearly demonstrate that fine-tuned Llama-3.1 not only outperforms its own base version and all evaluated GPT baselines, but also offers the additional advantages of data privacy and process transparency, making it particularly well-suited for this application.

Figure 9.6 depicts the loss values over training iterations. The model undergoes a significant reduction in loss during the initial stages of training, particularly within the first epoch, reflecting rapid early learning.

We further analyze the training dynamics of the fine-tuned GPT-4o-mini model by tracking

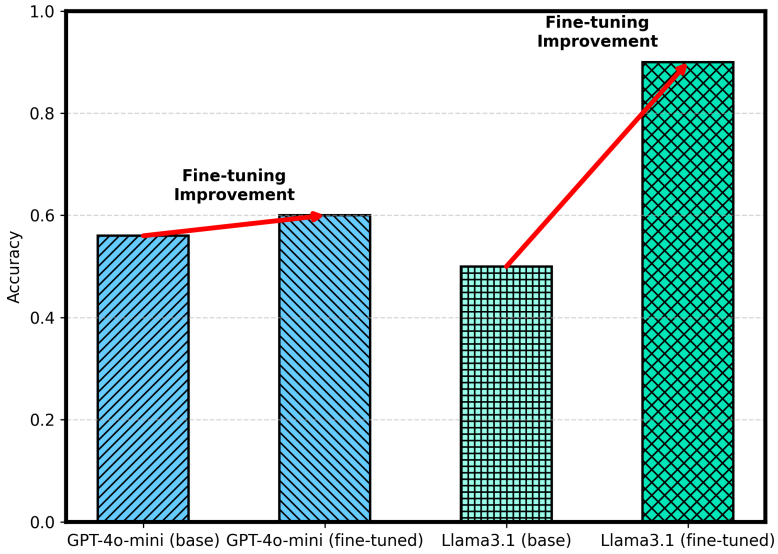


Figure 9.5 Comparing the accuracy of our models.

Table 9.4 Performance comparison on a subset of test samples.

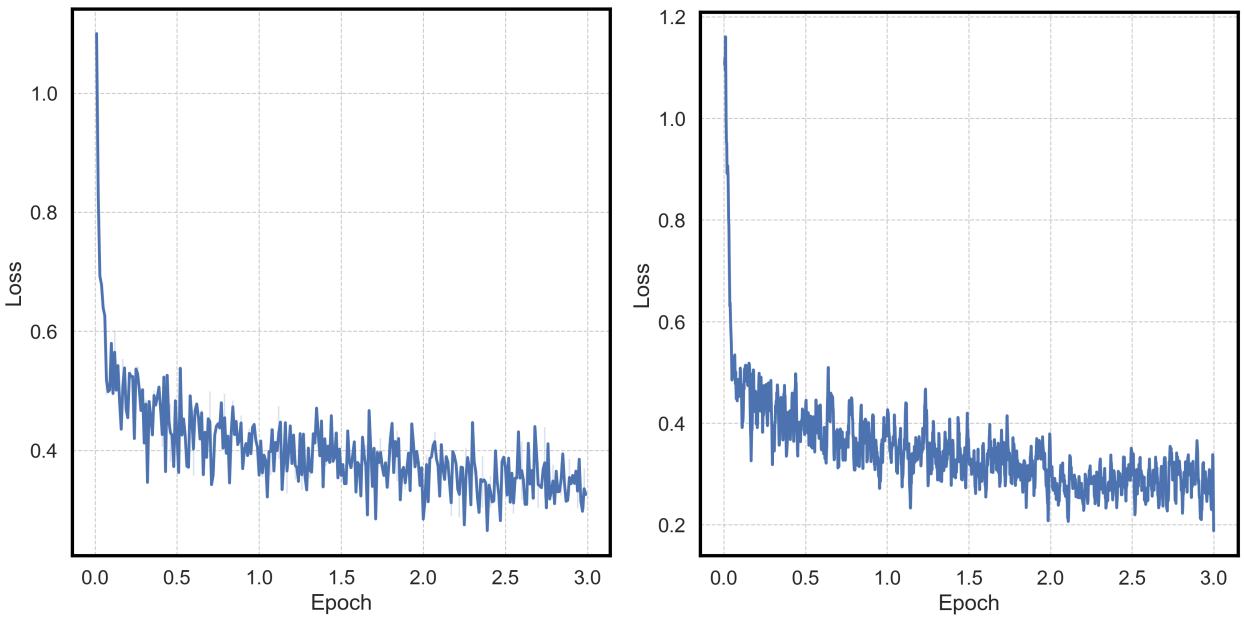
Metric-Models	FT.* Llama3.1	Base Llama 3.1	FT.* GPT-4o-mini	GPT-4o-mini	GPT-4.1-mini
Accuracy	**90.00**	50.00	60.00	56.00	76.00
Precision	**93.55**	60.60	63.04	60.86	77.77
Recall	**90.63**	63.16	90.62	87.50	87.50
F1-score	**92.06**	58.54	74.35	71.79	82.35
Cons.* Vul. Reason	**77.48**	51.03	75.89	55.45	62.16
Cons.* Sec. Risk	**68.86**	49.51	67.33	65.60	66.24
Sim.* Fixed Code	**87.25**	66.07	83.41	65.19	59.99

* **Abbreviations:** FR.: fine-tuned; Cons.: Consistency; Sim.: similarity; Vul.: Vulnerability; Sec.: Security.

the loss over training iterations. As shown in Figure 9.6b, the loss decreases rapidly during the early stages of training, suggesting that the model quickly learns task-specific patterns. However, despite this reduction in training loss, the final evaluation reveals that the model's accuracy does not improve accordingly.

Figure 9.7 illustrates the learning rate schedule throughout training. The learning rate steadily decays over iterations based on the cosine scheduler. This gradual reduction helps maintain learning stability and facilitates fine-tuning model parameters in later training stages.

We further analyze training throughput, measured as the number of samples processed per second, to evaluate computational efficiency. As shown in Figure 9.8, throughput remains largely stable throughout the training process, with only minor fluctuations. This consistency suggests efficient pipeline utilization. The absence of noticeable drops in throughput implies



(a) LLaMA 3.1 model (b) GPT-4o-mini

Figure 9.6 Loss versus epochs during fine-tuning: (a) LLaMA 3.1 model; (b) GPT-4o-mini.

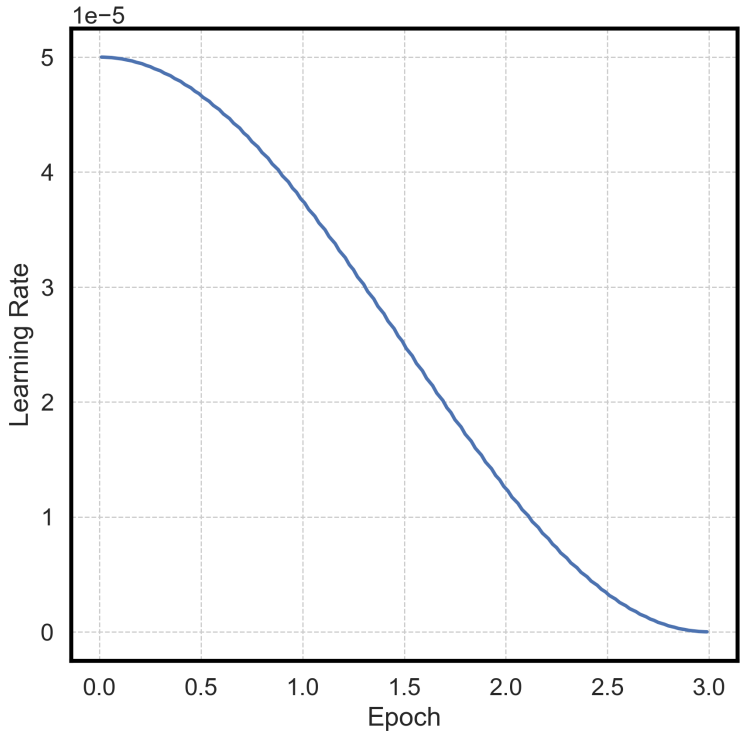


Figure 9.7 Learning VS epochs through fine-tuning the Llama 3.1 model.

that no significant memory, I/O, or processing bottlenecks were encountered. A slight upward trend in later epochs is attributed to system stabilization. Such stable throughput is crucial for large-scale training environments where performance scalability and resource efficiency are essential.

Regarding the performance of the developed plugin, we evaluate the practical utility and usability of our VSCode plugin, available on our GitHub repository [3] for vulnerability detection on solidity smart contracts containing three prevalent vulnerabilities: integer overflow/underflow, timestamp dependency, and reentrancy. The evaluation was conducted using VSCode version 1.79.0 with solidity plugin support and Node.js 18. This part of the analysis focuses on the plugin's detection accuracy, memory efficiency, and response time.

The plugin was tested across 27 known vulnerabilities distributed among multiple smart contracts. It successfully identified all 27 vulnerabilities, achieving a perfect detection rate with no false positives. For each identified vulnerability, the plugin provided: (i) highlighted vulnerable code lines, (ii) a detailed textual explanation outlining the cause and associated security risks, and (iii) a suggested fix in the form of replaceable Solidity code.

Compared to conventional static analysis tools such as Mythril and Slither, our approach offers several key advantages. It integrates seamlessly into the development environment, delivering real-time vulnerability detection, clear explanations, and actionable remediation suggestions. As shown in Table 9.5, traditional tools do not support interactive detection as well as contextual vulnerability reasoning. In contrast, our plugin supports both real-time and static vulnerability analysis.

The average memory usage during plugin execution was 62.39 MB, which shows the tool's lightweight and practical design. Overall, these results confirm the usability, accuracy, and integration potential of our plugin as a real-time auditing tool for enhancing the security of smart contract development.

9.6 Conclusion

This work presents a comprehensive approach to advancing smart contract vulnerability detection through dataset creation and model innovation. We curated and released a high-quality, multi-format dataset, including both contract-level and function-level samples, with rich, structured annotations to facilitate robust training and evaluation. Building upon this foundation, we fine-tuned the open-source *LLaMA-3.1-8B-Instruct* model using parameter-efficient strategies such as LoRA, achieving significant improvements in vulnerability detec-

[3] `https://github.com/erfan38/plugin.git`

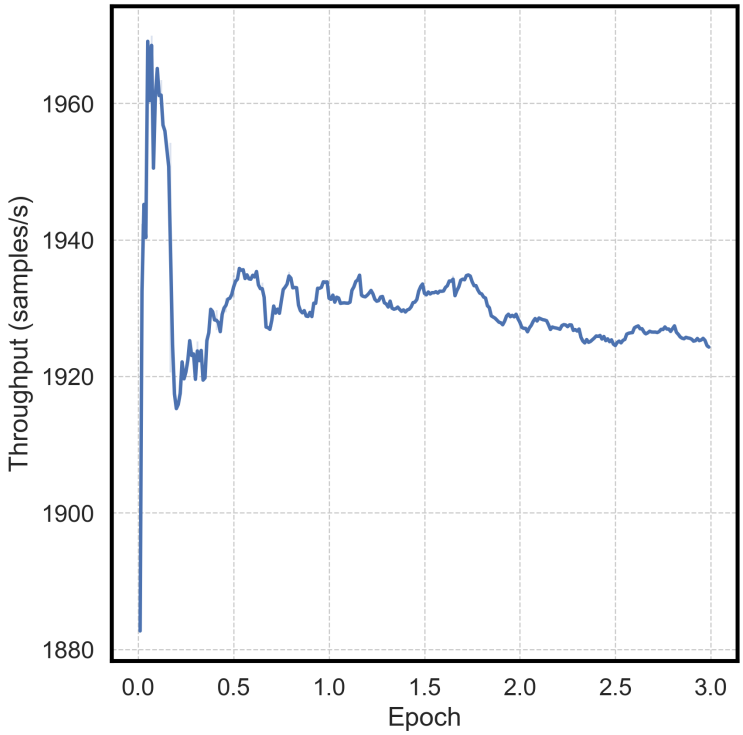


Figure 9.8 Throughput VS epochs through fine-tuning the Llama 3.1 model.

Table 9.5 Comparing the functionalities of existing vulnerability detection tools.

Tools/ Features	Detection	Reasoning	Code correction	Real-time
Our tool	✓	✓	✓	✓
Oyente [74]	✓	✗	✗	✗
Mythril [73]	✓	✗	✗	✗
SmartCheck [71]	✓	✗	✗	✗

tion accuracy, precision, recall, and interpretability compared to both the base LLaMA and closed-source models like GPT-4.1-mini and GPT-4o-mini (base and fine-tuned). Our evaluation, encompassing both classification and semantic metrics, demonstrates that our approach yields superior performance across all relevant dimensions.

In addition to the model contributions, we designed and implemented a VSCode plugin that enables real-time vulnerability detection and interactive remediation within the developer's workflow. The plugin integrates seamlessly with industry-standard development tools and provides actionable feedback, including highlighted code, vulnerability explanations, and fix suggestions, while maintaining low memory usage.

Overall, our work bridges the gap between API-dependent solutions and accessible, open-

source security auditing tools, paving the way for scalable, privacy-aware, and developer-friendly smart contract vulnerability detection in real-world blockchain ecosystems.

In the future, we aim to optimize our model for efficient, lightweight deployment on standard CPUs and edge devices. This will enable fully local, real-time vulnerability detection within development tools, reducing reliance on external cloud services and ensuring greater privacy and accessibility for developers in resource-constrained environments.

CHAPTER 10 GENERAL DISCUSSION

This chapter is organized into three sections: 1) How this thesis addresses the problems defined in Chapter 1, 2) How our solutions have proven superior to existing ones, and 3) The limitations and potential improvements of our work.

10.1 Addressing research objectives

The main objective of this thesis is to develop a set of security solutions to protect blockchain networks and applications against DoS vulnerabilities and attacks across multiple domains. This section provides a brief overview of the targeted problem statements defined in Chapter 1.12.

The **first problem** we identified pertains to the early detection of eclipse attacks in blockchain-based Metaverse systems. This challenge is comprehensively addressed in Chapter 4, where our proposed methodology is detailed. Consequently, the primary research objective—developing effective detection mechanisms for security attacks, including eclipse attacks—directly aligns with this problem and is fully addressed through our approach.

The **second problem** focused on designing a comprehensive framework to address selfish mining attacks within IoT networks. Two complementary approaches were proposed to tackle this issue: one for detection and another for mitigation. The detection approach, detailed in Chapter 5, leverages machine learning techniques, specifically Random Forest Classification (RFC) in conjunction with Principal Component Analysis (PCA), to effectively identify selfish mining behaviors. This methodological advancement fulfills the first part of the second research objective, namely the detection of security attacks such as selfish mining, and directly addresses the core of the second problem.

Furthermore, the thesis addresses the second aspect of this problem by introducing a novel mitigation strategy for security attacks, such as selfish mining, as described in Chapter 6. This mitigation approach models miner interactions as a war of attrition game, utilizing evolutionary game theory to examine the long-term stability of honest mining strategies. The framework is further enhanced by the introduction of dynamic punishment and reward mechanisms, as well as a reputation scoring system that penalizes miners for block withholding behavior. Simulation results confirm that these integrated mechanisms not only incentivize honest participation but also significantly reduce the success rate of selfish mining attacks. Consequently, the third research objective—corresponding to the mitigation component of

the second problem—has been successfully achieved.

The **third problem** addressed in this thesis involved the development of an effective scheme for the prevention of Sybil attacks. The approach detailed in Chapter 7 introduces a defense mechanism that integrates decentralized federated learning (DFL) with a secure, smart contract-based reputation system, thereby enabling both detection and prevention. Specifically, this methodology utilizes DFL to identify Sybil nodes by simulating a PoA-Ethereum-based IIoT network and generating a comprehensive dataset that captures the behavioral patterns of Sybil nodes using the proposed simulator. Within the DFL framework, each node independently trains a local model, exchanges model weights with neighboring nodes for aggregation, and conducts anomaly analysis to identify behaviors indicative of Sybil activity.

In addition, this thesis presents a practical simulator capable of modeling a blockchain-based network comprising multiple protocols, specifically designed to simulate Sybil attacks. As described in Chapter 7, the simulator supports a multi-protocol environment, incorporating ENR, MQTT, discovery, libp2p, and devp2p protocols. Consequently, the objective of addressing Sybil attacks—corresponding to the research problem—was successfully accomplished.

The **fourth problem** addressed in this thesis led to the development of a comprehensive framework for detecting smart contract vulnerabilities. The approach presented in Chapter 8 employs an open-source benchmark dataset to identify smart contracts susceptible to Integer Overflow/Underflow (IoU), Reentrancy (RE), and Timestamp Dependency (TD) vulnerabilities. Recognizing the potential biases introduced by bug injection, a thorough preprocessing phase was undertaken to construct an unbiased dataset. The proposed solution leverages both GPT-4o and GPT-4o-mini models, which not only detect vulnerabilities but also provide detailed reasoning, assess potential security risks, and suggest corresponding code fixes.

A further challenge tackled in this study was the lack of comprehensive, reliable datasets for smart contract vulnerability research—a critical barrier in this domain. As described in Chapter 9, we curated, refined, and released a high-quality, multi-format dataset, publicly available via GitHub[1], encompassing both contract-level and function-level samples. This dataset is richly annotated and structured to facilitate robust training and evaluation of detection models.

In addition, this thesis addresses the need for an effective, real-time tool to assist developers in producing secure smart contracts. Chapter 9 details the implementation of a VSCode plugin that provides real-time vulnerability detection and interactive remediation within the

[1] `https://github.com/erfan38/FT_LLM_SCVD.git`

developer workflow. This plugin seamlessly integrates with industry-standard development environments and delivers actionable feedback, including code highlighting, vulnerability explanations, and fix recommendations, all while maintaining low memory consumption.

Moreover, the thesis confronts the absence of a reliable, accurate, and publicly available large language model for smart contract vulnerability detection. The solution, described in Chapter 9, involves fine-tuning the open-source *LLaMA-3.1* model using parameter-efficient techniques such as LoRA. This approach achieves significant improvements in detection accuracy, precision, recall, and interpretability, outperforming both the base LLaMA and closed-source alternatives like GPT-4.1-mini and GPT-4o-mini (in both base and fine-tuned variants). Extensive evaluations using classification and semantic metrics confirm that our methodology delivers superior performance across all relevant criteria. In summary, this thesis fully achieves the fifth objective, thereby providing a comprehensive solution to the fourth problem.

10.2 Key strengths of the proposed solutions

This section provides a comparative analysis of our proposed approaches against existing state-of-the-art methods, highlighting the distinctive advantages and innovations introduced in this thesis.

10.2.1 Eclipse attack detection

Our community detection-based Intrusion Detection System (IDS), leveraging the Leiden algorithm, achieves robust detection with minimal resource overhead. In contrast to prior methods that rely on packet-level features or static classification models, our approach constructs a graph-based representation of network behavior and dynamically identifies malicious subgroups.

In our simulation, we emulate an eclipse attack by biasing the victim's peer address manager (*addrman*) so the *tried* table is dominated by attacker-controlled addresses from multiple independent IP groups (IPv4 /16). Following Heilman et al. [52], we assume $s = 32$ groups, a resource level shown to cover most tried buckets and reliably bias outbound selection, providing an effective and realistic adversary model.

Table 10.1 compares our approach with three notable works: Xu et al. [131], Dai et al. [133], and Alangot et al. [132]. While Xu and Dai [131] utilize random forest and CNN along with bidirectional long short-term memory (Bi-LSTM) models, with handcrafted or extracted features, their techniques incur significantly higher computational, communication, and storage overhead, making them less suitable for lightweight blockchain nodes. Alangot et al. [132]

Table 10.1 Comparison of Eclipse attack detection approaches.

Feature	**Ours**	**Xu et al. [8]**	**Dai et al. [7]**	**Alangot et al. [5]**
Detection Technique	CDA (Leiden)	RFC	CNN + Bi-LSTM	Gossip-Based Blockchain View Comparison
Client Computation Overhead	O(1) (Constant)	Moderate	High	Low
IDS Computation Overhead	O(n) (Linear)	O(n log n)	$O(n^2)$	O(n)
Storage Overhead	Low (on IDS)	High	High	Moderate
Communication Cost	Minimal (1 request + 1 response)	Continuous Packet Stream	Continuous Feature Stream	Periodic Blockchain Header Sync
Adaptability to Lightweight Nodes	High	Moderate	Low	Moderate
Real-time Detection	Yes	No	No	Partially
Scalability	High	Moderate	Low	Moderate

introduce a decentralized, gossip-based protocol that compares blockchain views, offering lightweight operation and strong scalability. Because synchronization occurs only at periodic intervals and hinges on the prompt propagation of block headers, the approach provides limited real-time detection. In addition, the threat model assumes an adversary capable of hijacking peer-to-peer connections while lacking the capacity to compromise multiple web servers simultaneously—an assumption that remains fragile. An attacker with sufficient resources and control over every network path to the victim could therefore undermine their proposed protocol.

Unlike these approaches, our IDS distributes complexity between the client and IDS server, enabling $O(1)$ client overhead and $O(n)$ IDS-side computation. It achieves strong detection performance without requiring intensive message parsing or deep packet inspection, thereby enhancing scalability and reducing false positives.

The time complexity of the proposed approach is $O(n)$. Let n denote the number of nodes per graph and $k = 8$ the constant number of message-specific graphs. For each graph $G(i)$, Leiden community detection runs in time near-linear in the number of edges:

$$T_{\text{Leiden}} = O\Big(\textstyle\sum_{i=1}^{k} m_i \Big).$$

Under sparsity (bounded average degree, hence $m_i = \Theta(n)$), we obtain $T_{\text{Leiden}} = O(n)$. For the k-way intersection of the community sets $L(i)$, we implement a hash-set–based intersec-

tion which yields expected $O(1)$ membership and insertion, giving

$$T_{\cap} = O\Big(\sum_{i=1}^{k} |L(i)| \Big) = O(n).$$

Therefore,

$$T_{\text{total}} = T_{\text{Leiden}} + T_{\cap} = O(n) + O(n) = O(n).$$

In summary, the algorithm runs in linear time per detection cycle, ensuring scalability with network size while preserving the $O(1)$ client overhead.

10.2.2 Selfish mining attack detection

We simulated a Bitcoin network and launched selfish mining attacks, gathering a dataset of 102,400 samples over 100 iterations for 16 full-node miners. Our lightweight detection framework, utilizing Principal Component Analysis (PCA) in conjunction with Random Forest Classification (RFC), achieves notable improvements in both accuracy and computational efficiency over the ForkDec deep learning approach. As illustrated in Table 10.2, our framework increases detection accuracy from 98.98% to 99.96% and reduces execution time by approximately 75% (from 442.88 seconds to 109.14 seconds). Furthermore, the RFC-based method is inherently less resource-intensive, making it particularly suitable for deployment in IoT environments where computational resources are limited.

Table 10.2 Comparison of proposed PCA+RFC method with ForkDec (FC-NN) for selfish mining detection

Metric	**ForkDec (FC-NN)**	**Proposed (PCA + RFC)**	**Improvement**
Accuracy (%)	98.98	99.96	+0.98
Recall (%)	100.00	99.76	–0.24 (negligible)
Precision (%)	95.86	100.00	+4.14
F1 Score (%)	97.88	99.88	+2.00
Execution Time (sec)	442.88	109.14	↓ ∼75% faster
Model Complexity	12-layer FC-NN	Lightweight RFC + PCA	Lower computation/storage
IoT Suitability	Limited	High	Deployable and efficient

10.2.3 Selfish mining attack mitigation

In contrast to prior studies that employ static incentives or solely rely on reactive countermeasures, our proposed retaliation game framework introduces a dynamic, adaptive strategy for discouraging selfish mining. Through the integration of evolutionary game theory, the framework incorporates time-sensitive punishment, decaying rewards, and reputation-based

deterrence, fostering the long-term dominance of honest mining strategies. Simulation results reveal stable convergence toward honest behavior, underscoring the framework's capacity for adaptive resilience.

To provide a broader perspective and deeper understanding of the field, our complementary work (see Appendix A) presents a comprehensive taxonomy of recent studies on the application of game-theoretic models in blockchain security. In that work, we systematically analyze the strengths, limitations, and contributions of existing approaches, highlighting how different game-theoretic frameworks have been employed to address various security threats. Through this analysis, we identified the retaliation game, specifically, a war of attrition formulation, as a particularly effective model for capturing the nuanced interactions between selfish and honest miners.

These insights directly informed the design of our dynamic mitigation framework in the present study, motivating our adoption of adaptive, reputation-based incentives to discourage selfish mining. By bridging the theoretical foundations from our taxonomy with our empirical results, we demonstrate the practicality and effectiveness of the retaliation (war of attrition) game model in fostering stable and honest participation within blockchain networks.

In our framework, concealment detection and reputation updates require no central authority. All nodes independently determine concealment by comparing a block's mining timestamp with its actual reception time and checking if it extends a previously unseen private chain—such timing gaps and sudden chain reveals indicate withholding. This mechanism also applies when honest miners temporarily withhold blocks in our retaliation strategy to raise attacker costs.

Using these public, on-chain events and a shared scoring formula, each node locally updates its reputation matrix to reflect positive scores for timely reveals and negative scores for concealment. To limit communication overhead, nodes only gossip compact updates or respond on demand, and may optionally publish small cryptographic summaries (e.g., Merkle roots) on-chain for universal verification. This ensures fairness, decentralization, and efficiency in maintaining the reputation system.

Table 10.3 presents a comparison of leading game-theoretic approaches for selfish mining mitigation. Unlike previous models, which are grounded in static assumptions or fixed reactive rules, our framework uniquely accommodates dynamic adaptation to evolving network conditions and miner strategies. By achieving stable equilibria that favor honest participation, the proposed approach establishes a practical and sustainable mechanism for enhancing blockchain security.

Table 10.3 Comparison of game-theoretic approaches for selfish mining mitigation

Eyal et al. (2015) [148]	Non-cooperative game	Attack profitability requires $\geq 33\%$ hash power	Static
Luu et al. (2015) [142]	Power splitting game	Pools strategically split hash power	Static
Zhen et al. (2017) [143]	Zero-Determinant strategy	Repeated interactions, payoff pinning	Static
Sun et al. (2022) [150]	Tit-for-Tat	Mirrors opponent's actions	Static
Our solution (2025)	War of Attrition	Adaptive reward, punishment, reputation; honest miner stability	Dynamic

10.2.4 Sybil attack prevention

We introduce a DFL framework for Sybil attack detection, designed to preserve data privacy by eliminating the need to share raw data between nodes. Relative to centralized learning-based approaches, our solution demonstrates enhanced privacy, scalability, and robustness. The integration of a smart contract-based reputation mechanism further strengthens trust by penalizing malicious brokers through on-chain behavioral analysis. The prevention mechanism internally scales raw anomaly scores into the discrete 0–10 severity range, normalizes them across validators for consistent interpretation, and applies context-aware adjustments (e.g., detection confidence, reporter reputation, evidence age) to ensure penalties remain proportionate, fair, and robust.

The comprehensive evaluation of our Sybil defense framework was conducted within a simulated PoA-MQTT-based IIoT network, capturing communications across MQTT and Ethereum-related protocols (ENR, discovery, libp2p, devp2p). As detailed in Chpater 7, experimental results highlight that our DFL-based detection approach not only achieves strong predictive performance but also addresses critical limitations of prior methods, including reliance on centralized servers and associated privacy risks.

Notably, compared to conventional federated learning, our framework attains higher and more consistent F1-scores across all clients (see Table 10.5), demonstrating its ability to sustain balanced detection performance regardless of network partitioning or client heterogeneity. While centralized deep learning approaches may attain marginally higher accuracy, their susceptibility to single points of failure and privacy vulnerabilities makes them unsuitable for trust-sensitive decentralized systems.

Moreover, as indicated in Table 10.4, our framework uniquely offers high Sybil resistance,

model tamper protection, and scalable trust—all features not concurrently provided by existing DL or FL methods.

10.2.5 Smart contract vulnerability detection

Our use of GPT-4o with chain-of-thought (CoT) prompting surpasses both traditional static and dynamic analysis tools and recent LLM-based methods. Unlike existing tools, our approach not only achieves higher recall and fewer false positives but also provides explainable vulnerability insights through line-level detection, reasoning, and code correction. As shown in Table 10.6, compared to the recent state-of-the-art approach by Du et al. [5], our method performs better across all targeted vulnerability types, including reentrancy, integer overflow/underflow, and timestamp dependency. Moreover, our work contains code fixing as well as providing the lines of vulnerable code. The detailed analysis of our approach is stated in Chapter 8.

10.2.6 Fine-tuning LLMs for detection of smart contract vulnerabilities

Chapter 9 introduces an open-source solution for smart contract vulnerability detection by fine-tuning the Llama 3.1 model. This builds upon our previous work (Chapter 8), where GPT-based APIs were leveraged for vulnerability detection. While the GPT-API-based solution attained strong performance metrics, its reliance on proprietary APIs poses limitations regarding data privacy, cost, and reproducibility. In contrast, the approach presented in Chapter 9 overcomes these issues by enabling local, transparent fine-tuning on the Llama model.

The empirical results presented in Chapter 9 clearly demonstrate that fine-tuning Llama-3.1 leads to a substantial improvement in all evaluation metrics compared to the non-fine-tuned (base) Llama model. Specifically, the fine-tuned Llama3.1 achieves an accuracy of 90%, significantly surpassing its base counterpart, which only reached 50%. These gains extend across other critical metrics, including precision (93.55%), recall (90.63%), and F1-score (92.06%)—as well as semantic alignment metrics that capture the quality of vulnerability reasoning and fixed code generation. The superiority of the fine-tuned Llama model is also evident when compared to both fine-tuned and base versions of GPT-4o-mini and the newly released GPT-4.1-mini, which did not achieve comparable improvements despite their advanced architectures.

A major advantage of our approach lies in its open-source and locally deployable nature, which enables secure handling of sensitive smart contract datasets and eliminates concerns regarding

Table 10.4 Comparative analysis of DL, FL, and DFL across key performance criteria.

Criteria	DL	FL	DFL
Data Privacy	No	Partially	Yes
Single Point of Failure	No	No	Yes
Sybil Attack Resistance	Low	Low	High
Model Tamper Protection	Low	Low	High
Trust Scalability	Low	Limited	High
Time complexity	Low	Moderate	High
Communication overhead	Low	Moderate	High

Explanation: *Low*, *Limited*, and *Moderate* denote minimal, restricted, and intermediate capabilities, respectively; *High* indicates strong capability. Criteria definitions are detailed in the text.

Table 10.5 Comparing evaluation metrics across DL, FL, and DFL (including per-client results)

Metrics (%)	DL	FL Clients					DFL Clients				
		Client 1	Client 2	Client 3	Client 4	Client 5	Client 1	Client 2	Client 3	Client 4	Client 5
Accuracy	93.46	87.08	86.98	86.79	89.30	86.62	89.57	90.61	91.40	90.60	89.93
Precision	93.51	86.94	86.88	86.94	89.44	86.55	91.74	92.77	92.77	92.76	91.58
Recall	93.46	86.97	87.00	86.67	89.42	86.62	89.80	90.59	91.63	90.31	89.11
F1-score	93.48	86.95	86.93	86.67	89.43	86.55	89.77	90.86	91.52	90.71	89.25

third-party data exposure. This is particularly vital for security-critical applications, as it ensures both data privacy and full control over the training process. In addition, the observed improvements in semantic and text-based evaluation metrics (see Table 10.7 and Table 10.8) confirm that our model not only excels in classification tasks but also generates high-quality, contextually relevant vulnerability explanations and code fixes.

Table 10.7 Evaluation of generated text alignment with reference labels across 1,628 test samples.

Metric	Score (%)
Vulnerability Reason	78.12
Potential Security Risk	73.69
Fixed Code	89.76
Vulnerable Code	95.14

While proprietary LLMs such as GPT-4o-mini offer impressive results, our findings indicate that open-source models like Llama 3.1 can be tailored, via fine-tuning (FT), to outperform or match closed-source alternatives, all while offering greater transparency and reproducibility. This democratizes access to high-performance smart contract security tools and empowers organizations to deploy robust, privacy-preserving detection systems without dependency on

Table 10.6 Comparison of Smart Contract vulnerability detection approaches.

Metric	**Our Approach (GPT-4o)**	**Du et al. [5]**
IoU Precision (%)	92.2	85.7
IoU Recall (%)	93.5	75.4
RE Precision (%)	98.1	91.3
RE Recall (%)	95.4	88.2
TD Precision (%)	93.4	84.5
TD Recall (%)	93.8	79.1
Code Reasoning	✓	✓
Code Fixing	✓	✗
False Positives	Low	Low
False Negatives	Low	Moderate
Line-level Detection	✓	✗

Note: Du et al. [5] represent a recent LLM-based approach. Traditional tools are excluded from this table due to differences in evaluation scope and dataset compatibility.

external vendors.

The results validate the effectiveness of fine-tuning open-source LLMs for security-critical tasks, bridging the performance gap with proprietary solutions while delivering substantial benefits in transparency, privacy, and reproducibility. These contributions mark a significant step toward the broader adoption of LLM-driven security frameworks in decentralized blockchain ecosystems.

10.3 Limitations and future improvements

In this section, we discuss the limitations of each proposed approach and the possible future directions of improvement.

10.3.1 Eclipse attack on Metaverse

Despite designing an IDS for detecting eclipse attacks in Metaverse networks, some limitations can be considered as future directions for researchers. This framework depends on a centralized intrusion detection system, which is a key limitation of this work. The single point of failure may occur and could cause a bottleneck, particularly in dynamic and decentralized systems like the Metaverse, and may bring the system down. Therefore, this architecture lacks autonomous detection capabilities for individual blockchain nodes, especially light or null nodes with resource constraints. One idea can be using a distributed IDS that works with

Table 10.8 Comparison table on subset of test samples.

Metric \ Models	FT. Llama3.1	Base Llama 3.1	FT. GPT-4o-mini	GPT-4o-mini	GPT-4.1-mini
Accuracy	**90.00**	50.00	60.00	56.00	76.00
Precision	**93.55**	60.60	63.04	60.86	77.77
Recall	**90.63**	63.16	90.62	87.50	87.50
F1-score	**92.06**	58.54	74.35	71.79	82.35
Consistency Vul. Reason	**77.48**	51.03	75.89	55.45	62.16
Consistency Sec. Risk	**68.86**	49.51	67.33	65.60	66.24
Similarity Fixed Code	**87.25**	66.07	83.41	65.19	59.99

the nature of blockchain; each node can participate in intrusion detection, but a verification setup will be necessary to avoid malicious activities.

Another limitation of this work is that, despite high accuracy, precision, and F1, the recall is comparatively lower, implying that a non-negligible fraction of true attacks goes undetected. This shortfall primarily arises from (i) a strict all-layer intersection that is overly conservative, (ii) unstable community assignments in sparse or time-varying graphs, (iii) sensitivity to the Leiden resolution parameter (merging or fragmenting small attacker communities), and (iv) behavioral mimicry whereby attackers resemble benign peers in specific protocols.

We deliberately optimized for precision to minimize false positives, which in a metaverse setting can disconnect legitimate users, degrade the quality of experience, and trigger costly recovery actions (e.g., peer churn, state resynchronization). This precision-first design reduces collateral damage and operational risk, but it also increases false negatives (i.e., lowers recall). As future work, this limitation may be addressed by integrating the approach with machine learning techniques or by employing a collaborative IDS in which detection tasks are shared among multiple nodes.

10.3.2 Detection of selfish mining attack

Despite its strong performance and demonstrated superiority over existing state-of-the-art methods, this work is subject to two limitations. A primary challenge in the study of security attacks on blockchain-based networks lies in the generation of comprehensive, real-time datasets. Similar to several related works, this work relies on two datasets: one generated using a widely adopted simulator and the other sourced from previous state-of-the-art research. However, real-world blockchain environments exhibit unpredictable behaviors, diverse miner strategies, and various forms of noise that may not be fully captured through simulation-based datasets.

Another limitation of this work is that it assumes a single selfish miner or pool, while ne-

glecting other assumptions closer to real-time networks, such as semi-selfish miners or groups of colluding miners.

For future work, it is recommended to develop a comprehensive dataset encompassing both benign and selfish network behaviors, which should be made publicly accessible to the research community. Additionally, deploying and evaluating the proposed detection approach on public testnets or mainnets would enable the collection and analysis of real network traffic and miner behaviors across a variety of unforeseen conditions. Furthermore, extending the threat model to include scenarios with multiple colluding selfish miners, semi-selfish actors, or adaptive adversaries employing dynamic strategies would offer a more thorough assessment of the robustness of the detection framework in real-world environments.

10.3.3 Mitigation of selfish mining attack

While the proposed framework successfully mitigates selfish mining attacks through the integration of game-theoretic incentive mechanisms, several aspects remain open for further enhancement. A key limitation of the current study lies in its abstraction from concrete real-world network conditions. Specifically, factors such as network latency, block propagation time, mining power distribution, and P2P topology are not explicitly modeled in the present assumptions. These elements can have a substantial impact on the effectiveness, profitability, and stability of selfish mining and countermeasures in operational blockchain environments.

Furthermore, the evaluation of the proposed model is limited to numerical simulations and theoretical analyses. Although these results provide valuable insights into the dynamics and equilibrium states of miner behavior, the absence of empirical validation in large-scale testnet environments leaves open questions regarding the scalability and resilience of the approach under realistic conditions.

As a promising avenue for future research, it is recommended that this work be extended by implementing the proposed mechanisms in simulation testbeds that more accurately reflect operational parameters. Incorporating network characteristics such as latency, block propagation delays, dynamic mining power allocation, and varying peer connectivity will allow for a more comprehensive assessment of the model's effectiveness and adaptability. Such efforts will contribute to developing more robust and scalable solutions for enhancing blockchain security against selfish mining attacks.

10.3.4 Prevention of Sybil attacks

Despite the demonstrated advantages of this work over existing approaches, some limitations remain. First, the computational and communication overhead introduced by the DFL framework is not negligible, as DFL framework requires high communication capacity across the network. This could restrict the scalability of the solution as it demands a powerful communication infrastructure. As a future direction, it is recommended to explore techniques for reducing both computational and communication overhead, thereby enhancing the efficiency and scalability of the proposed approach.

While the proposed reputation management mechanism represents a step toward automated Sybil prevention, the next step to further improve this work is to deploy and empirically validate it as a smart contract within a real blockchain-based system.

As a future direction, it is recommended that the proposed approach be implemented and evaluated in a practical Ethereum-MQTT IIoT testnet. This would allow for the collection and analysis of real-world data, validation of the reputation management system, and assessment of the solution's robustness under realistic network conditions.

Sybil simulator

The Sybil attack simulator developed in this thesis offers several key advantages. It supports a multi-protocol environment by emulating diverse communication channels, such as MQTT, devp2p, libp2p, discovery, and ENR, closely reflecting the protocol diversity of Ethereum-integrated IIoT networks. The simulator enables the generation of rich, labeled datasets by modeling both high-risk and low-risk Sybil node behaviors, along with multiple attack strategies (spammer, flooder, evasive). It also provides extensive behavioral feature extraction, facilitating rigorous experimentation with machine learning-based detection and prevention frameworks. The tool's configurability, including adjustable network size and attack intensity, makes it suitable for a variety of experimental scenarios.

However, several limitations remain. The simulator abstracts some aspects of real-world network operation; for instance, it does not model actual packet-level exchanges, P2P network topologies, or protocol stack implementations. All interactions are synthetic, and adversarial behaviors are static rather than adaptive. Furthermore, the simulator operates in-memory, potentially limiting scalability for extensive network experiments. Future improvements could include integrating the simulator with testnets, modeling more sophisticated and adaptive Sybil attack strategies, implementing more realistic network and adversarial dynamics, and empirically validating the generated datasets against real-world data. These enhancements

would further increase the simulator's realism, scalability, and value for blockchain security research.

10.3.5 Detection of smart contract vulnerabilities

Despite the strong performance demonstrated by our proposed approach presented in Chapter 8, several challenges remain. First, the approach relies on the GPT-API, which introduces substantial cost considerations. Additionally, the accessibility of our method is contingent upon the availability of the external API; if the model becomes inaccessible, the approach cannot be utilized.

Furthermore, although the dataset used is carefully cleaned and normalized, it remains relatively small and is based on bug injections into real-world smart contracts. While these steps improve the dataset's quality, the limited size and synthetic nature of the bugs may affect the generalizability of the results.

To address these three limitations, we introduce fine-tuned large language models in Chapter 9, which mitigate reliance on external APIs and provide a more cost-effective and accessible solution. However, another remaining challenge is the phenomenon of LLM hallucinations, where the model may generate incorrect or misleading outputs. Addressing hallucination remains an important direction for future research.

Fine-tuning LLMs

The paper explained in Chapter 9 presents several significant contributions, including the creation of a comprehensive dataset, the implementation of a real-time VSCode plugin, and the fine-tuning of LLMs for smart contract vulnerability detection. Despite these advances, several opportunities remain for further enhancement.

One important future direction is the deployment of the proposed approach in real-world environments. Efficient deployment can be facilitated by adopting model optimization techniques, which support lightweight and resource-efficient inference. While this study leverages LoRA for parameter-efficient fine-tuning, allowing the model to update only a small subset of parameters instead of the full model, there is still room to improve deployment efficiency further and reduce resource consumption.

Another limitation is that the current dataset focuses on the top three vulnerabilities, including IoU, RE, and TD. Expanding the dataset to include additional emerging smart contract vulnerabilities would enhance the practical impact of the approach.

Therefore, it is recommended that future research focus on optimizing model deployment for greater efficiency and scalability and extending the dataset to cover a broader spectrum of vulnerabilities.

10.4 Conclusion

This chapter provided a comprehensive discussion of the thesis contributions by addressing the core research objectives, demonstrating the superiority of the proposed methods over existing solutions, and critically examining the limitations and potential avenues for future directions. Through detailed comparative analysis, the chapter highlighted how each proposed approach advances the state of the art in securing blockchain-based systems against attacks and smart contract vulnerabilities. Furthermore, it identified current challenges and outlined directions for future research to further improve the scalability and deployment of these solutions. This holistic discussion underscores the significance and impact of the study while laying a foundation for continued advancements in the field.

CHAPTER 11 CONCLUSION

This thesis presented a comprehensive set of approaches to advance the security of blockchain-based systems, addressing several pressing research problems spanning network attacks and smart contract vulnerabilities, mainly focusing on Denial of Service (DoS) threats. By formulating targeted objectives for each problem domain, the proposed methodologies offer significant advancements in detection, mitigation, and prevention, bridging theoretical innovations with practical solutions for real-world deployments.

The thesis began by tackling the challenge of early detection of eclipse attacks in blockchain-based Metaverse environments. Leveraging community detection algorithms, particularly the Leiden method, the proposed intrusion detection system demonstrated high precision, specificity, and minimal overhead, offering a practical solution that outperforms existing static rule-based and machine learning approaches in resource-constrained Metaverse scenarios.

Subsequently, this work addressed selfish mining attacks from two complementary perspectives: detection and mitigation. A lightweight machine learning framework employing principal component analysis and random forest classification was introduced for detection, improving accuracy and computational efficiency compared to current deep learning methods. For mitigation, we developed a novel incentive scheme by modeling miner interactions as a retaliation game, specifically, a war of attrition, to capture the strategic behavior of selfish and honest miners. We then applied evolutionary game theory to analyze the long-term stability and convergence of honest mining strategies under this framework.

To provide a broader perspective and deeper understanding, we also presented a comprehensive taxonomy of recent studies on the application of game-theoretic models in blockchain-based security, analyzing and explaining the strengths, limitations, and contributions of each relevant work.

Another contribution of this thesis is the robust defense against Sybil attacks in IIoT and blockchain environments. The integration of decentralized federated learning with a smart contract-based reputation management system established a dual defense: local, privacy-preserving detection and automated, on-chain prevention. Experiments using a custom PoA-Ethereum IIoT network simulator validated the scalability of the approach, resilience, and performance superiority over both centralized and conventional federated learning models.

The thesis also addressed the increasing risks posed by vulnerabilities in Ethereum smart contracts, which underpin decentralized applications. Two main solutions were proposed.

The first leveraged the latest advancements in large language models, including GPT-4o and GPT-4o-mini, to deliver accurate and explainable vulnerability detection, capable of providing line-level vulnerability explanations, risk reasoning, and code fixes. The second solution overcame the limitations of proprietary LLMs by fine-tuning an open-source LLaMA-3.1 model using parameter-efficient methods. This approach achieved comparable and superior results to other models, offering transparency and cost-effectiveness.

In addition, this thesis made several practical contributions to the research community:

- Releasing a comprehensive dataset of annotated smart contract vulnerabilities, which is now publicly available to support further advances in this domain.
- Developing a real-time Visual Studio Code (VSCode) plugin for smart contract vulnerability detection, providing actionable feedback and remediation suggestions within the developer workflow.
- Designing and implementing a simulator for Sybil attack scenarios in Ethereum-based IIoT networks, supporting empirical validation and benchmarking.

Despite the demonstrated superiority of the proposed frameworks, their adaptability and effectiveness in real-world, highly dynamic scenarios remain to be fully validated. In particular, empirical studies on public blockchains, live IIoT deployments, and under evolving adversarial strategies will be essential to comprehensively assess practical limitations, ensure resilience, and further enhance the scalability of these solutions. Therefore, several challenges and opportunities remain open for future research:

- **Decentralization of Detection:** Further research should explore fully decentralized intrusion detection architectures for eclipse attacks, enabling each blockchain node, especially lightweight nodes, to autonomously participate in detection and mitigation.
- **Real-World Validation:** Expanding the empirical evaluation of detection and mitigation schemes to public testnets, mainnets, and large-scale real-world deployments will provide deeper insights into practical limitations and edge-case scenarios.
- **Advanced Threat Models:** Extending threat models to include adaptive, colluding, and semi-selfish adversaries will offer more comprehensive security guarantees.
- **Model Optimization and Deployment:** Enhancing LLM deployment efficiency via model compression, quantization, or new inference architectures will broaden adoption in embedded and edge devices.

- **Expanding Dataset Scope:** Growing the coverage of annotated datasets to include emerging and less-studied vulnerabilities will strengthen generalization and future-proof automated detection.
- **Mitigating LLM Limitations:** Addressing issues such as hallucination and explainability in LLM-based analysis will be critical to achieving fully reliable automation.
- **Cross-Disciplinary Integration:** Combining blockchain security methods with privacy-preserving computation and trusted hardware represents a promising avenue for holistic defense mechanisms.

In summary, this thesis advances the field of blockchain and smart contract security through secure frameworks that integrate novel detection, mitigation, and prevention strategies, supported by open datasets and tools. The presented work lays a robust foundation for ongoing innovation and collaboration in the secure, scalable, and transparent design of blockchain-enabled infrastructure.

REFERENCES

[1] P. Qian *et al.*, "Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2220–2229.

[2] Y. Zhuang *et al.*, "Smart contract vulnerability detection using graph neural networks," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3283–3290.

[3] J. F. Ferreira *et al.*, "Smartbugs: A framework to analyze solidity smart contracts," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, 2020, pp. 1349–1352.

[4] F. Erfan *et al.*, "Advanced smart contract vulnerability detection using large language models," in *2024 8th Cyber Security in Networking Conference (CSNet)*. IEEE, 2024, pp. 289–296.

[5] Y. Du and X. Tang, "Evaluation of chatgpt's smart contract auditing capabilities based on chain of thought," *arXiv preprint arXiv:2402.12023*, 2024.

[6] Z. Wei *et al.*, "Llm-smartaudit: Advanced smart contract vulnerability detection," *arXiv preprint arXiv:2410.09381*, 2024.

[7] H. Wu *et al.*, "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 378–389.

[8] X. Li and W. Wu, "Recent advances of blockchain and its applications," *Journal of Social Computing*, vol. 3, no. 4, pp. 363–394, 2022.

[9] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE internet of things journal*, vol. 5, no. 2, pp. 1184–1195, 2018.

[10] M. Zhaofeng *et al.*, "A blockchain-based trusted data management scheme in edge computing," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2013–2021, 2019.

[11] Z. Bao *et al.*, "Iotchain: A three-tier blockchain-based iot security architecture," *arXiv preprint arXiv:1806.02008*, 2018.

[12] G. D. Putra *et al.*, "Trust management in decentralized iot access control system," in *2020 IEEE international conference on blockchain and cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–9.

[13] J. Leng *et al.*, "Secure blockchain middleware for decentralized iiot towards industry 5.0: A review of architecture, enablers, challenges, and directions," *Machines*, vol. 10, no. 10, p. 858, 2022.

[14] N. Dhieb *et al.*, "Scalable and secure architecture for distributed iot systems," in *2020 IEEE Technology & Engineering Management Conference (TEMSCON)*. IEEE, 2020, pp. 1–6.

[15] H. Guo and X. Yu, "A survey on blockchain technology and its security," *Blockchain: research and applications*, vol. 3, no. 2, p. 100067, 2022.

[16] E. Politou *et al.*, "Blockchain mutability: Challenges and proposed solutions," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1972–1986, 2019.

[17] X. Li *et al.*, "A survey on the security of blockchain systems," *Future generation computer systems*, vol. 107, pp. 841–853, 2020.

[18] M. Mirkin *et al.*, "Bdos: Blockchain denial-of-service," in *Proceedings of the 2020 ACM SIGSAC conference on Computer and Communications Security*, 2020, pp. 601–619.

[19] P. Tantikul and S. Ngamsuriyaroj, "Exploring vulnerabilities in solidity smart contract." in *ICISSP*, 2020, pp. 317–324.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

[21] R. Yang *et al.*, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1508–1532, 2019.

[22] M. A. Ferrag and L. Shu, "The performance evaluation of blockchain-based security and privacy systems for the internet of things: A tutorial," *IEEE Internet of Things Journal*, 2021.

[23] A. K. Jain, N. Gupta, and B. B. Gupta, "A survey on scalable consensus algorithms for blockchain technology," *Cyber Security and Applications*, vol. 3, p. 100065, 2025.

[24] J. Neu, E. N. Tas, and D. Tse, "Two more attacks on proof-of-stake ghost/ethereum," in *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, 2022, pp. 43–52.

[25] M. S. Ali *et al.*, "Applications of blockchains in the internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2018.

[26] M. Salimitari and M. Chatterjee, "A survey on consensus protocols in blockchain for iot networks," *arXiv preprint arXiv:1809.05613*, 2018.

[27] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *Journal of Information processing systems*, vol. 14, no. 1, pp. 101–128, 2018.

[28] Parity Technologies, "Proof of authority," https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains, 2017, accessed: 2025-04-15.

[29] S. De Angelis *et al.*, "Pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain," in *CEUR workshop proceedings*, vol. 2058. CEUR-WS, 2018.

[30] L. M. Bach, B. Mihaljevic, and M. Zagar, "Comparative analysis of blockchain consensus algorithms," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Ieee, 2018, pp. 1545–1550.

[31] Z. Zheng *et al.*, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE international congress on big data (BigData congress)*. Ieee, 2017, pp. 557–564.

[32] A. Kumar *et al.*, "Lightweight proof of game (lpog): a proof of work (pow)'s extended lightweight consensus algorithm for wearable kidneys," *Sensors*, vol. 20, no. 10, p. 2868, 2020.

[33] C. T. Nguyen *et al.*, "Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities," *IEEE access*, vol. 7, pp. 85 727–85 745, 2019.

[34] R. Yang *et al.*, "Assessing blockchain selfish mining in an imperfect network: Honest and selfish miner viewsf," *Computers & Security*, vol. 97, p. 101956, 2020.

[35] I. Homoliak *et al.*, "The security reference architecture for blockchains: Toward a standardized model for studying vulnerabilities, threats, and defenses," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 341–390, 2020.

[36] A. Reyna *et al.*, "On blockchain and its integration with IoT. challenges and opportunities," *Future generation computer systems*, vol. 88, pp. 173–190, 2018.

[37] Z. Zheng *et al.*, "Blockchain challenges and opportunities: A survey," *International journal of web and grid services*, vol. 14, no. 4, pp. 352–375, 2018.

[38] X. Wang *et al.*, "Survey on blockchain for internet of things," *Computer Communications*, vol. 136, pp. 10–29, 2019.

[39] H. F. Atlam *et al.*, "Blockchain with internet of things: Benefits, challenges, and future directions," *International Journal of Intelligent Systems and Applications*, vol. 10, no. 6, pp. 40–48, 2018.

[40] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.

[41] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.

[42] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, 2022, pp. 365–390.

[43] F. A. Alaba *et al.*, "Internet of Things security: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 10–28, 2017.

[44] A. R. Sfar *et al.*, "A roadmap for security challenges in the Internet of Things," *Digital Communications and Networks*, vol. 4, no. 2, pp. 118–137, 2018.

[45] S. Singh, A. S. Hosen, and B. Yoon, "Blockchain security attacks, challenges, and solutions for the future distributed iot network," *IEEE Access*, vol. 9, pp. 13 938–13 959, 2021.

[46] S. Zhao, S. Li, and Y. Yao, "Blockchain enabled industrial internet of things technology," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 6, pp. 1442–1453, 2019.

[47] Y. Wang *et al.*, "A survey on metaverse: Fundamentals, security, and privacy," *IEEE Communications Surveys & Tutorials*, 2022.

[48] T. A. Jaber, "Security risks of the metaverse world." *International Journal of Interactive Mobile Technologies*, vol. 16, no. 13, 2022.

[49] S. Raval, *Decentralized applications: harnessing Bitcoin's blockchain technology.* " O'Reilly Media, Inc.", 2016.

[50] S. K. Panda and S. C. Satapathy, "An investigation into smart contract deployment on ethereum platform using web3. js and solidity using blockchain," in *Data Engineering and Intelligent Computing: Proceedings of ICICC 2020.* Springer, 2021, pp. 549–561.

[51] W. Cai *et al.*, "Decentralized applications: The blockchain-empowered software system," *IEEE access*, vol. 6, pp. 53 019–53 033, 2018.

[52] E. Heilman *et al.*, "Eclipse attacks on {Bitcoin's}{peer-to-peer} network," in *24th USENIX security symposium (USENIX security 15)*, 2015, pp. 129–144.

[53] A. E. Yves-Christian *et al.*, "Total eclipse: How to completely isolate a bitcoin peer," in *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC).* IEEE, 2018, pp. 1–7.

[54] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Communications of the ACM*, vol. 61, no. 7, pp. 95–102, 2018.

[55] C. Grunspan and R. Pérez-Marco, "On profitability of selfish mining," *arXiv preprint arXiv:1805.08281*, 2018.

[56] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1.* Springer, 2002, pp. 251–260.

[57] P. Swathi, C. Modi, and D. Patel, "Preventing sybil attack in blockchain using distributed behavior monitoring of miners," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT).* IEEE, 2019, pp. 1–6.

[58] N. Szabo, "The idea of smart contracts. nick szabo's papers and concise tutorials," *URL http://www. fon. hum. uva. nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo. be*, 2018.

[59] M. Ren *et al.*, "Empirical evaluation of smart contract testing: What is the best choice?" in *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*, 2021, pp. 566–579.

[60] Z. Zheng *et al.*, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[61] C. Finance. (2023) C.r.e.a.m. finance, a decentralized lending protocol. Accessed: 2023-08-01. [Online]. Available: https://cream.finance/

[62] Z. Liu *et al.*, "Rethinking smart contract fuzzing: Fuzzing with invocation ordering and important branch revisiting," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1237–1251, 2023.

[63] Z. He, Z. Li, and S. Yang, "Large language models for blockchain security: A systematic literature review," *arXiv preprint arXiv:2403.14280*, 2024.

[64] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 906–917.

[65] F. Erfan, M. Bellaiche, and T. Halabi, "Game-theoretic Designs for Blockchain-based IoT: Taxonomy and Research Directions," in *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2022, pp. 27–37.

[66] M. Saad *et al.*, "Countering selfish mining in blockchains," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 360–364.

[67] F. Erfan, M. Bellaiche, and T. Halabi, "Community detection algorithm for mitigating eclipse attacks on blockchain-enabled metaverse," in *2023 IEEE International Conference on Metaverse Computing, Networking and Applications (MetaCom)*. IEEE, 2023, pp. 403–407.

[68] M. Iqbal and R. Matulevičius, "Exploring sybil and double-spending risks in blockchain systems," *IEEE Access*, vol. 9, pp. 76 153–76 177, 2021.

[69] Z. Liu *et al.*, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1296–1310, 2021.

[70] S. S. Kushwaha *et al.*, "Ethereum smart contract analysis tools: A systematic review," *Ieee Access*, vol. 10, pp. 57 037–57 062, 2022.

[71] S. Tikhomirov *et al.*, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain*, 2018, pp. 9–16.

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

[73] ConsenSys, "Mythril," 2018. [Online]. Available: https://github.com/ConsenSys/mythril

[74] L. Luu *et al.*, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[75] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proceedings of the 34th annual computer security applications conference*, 2018, pp. 664–676.

[76] C. Shou, S. Tan, and K. Sen, "Ityfuzz: Snapshot-based fuzzer for smart contract," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 322–333.

[77] P. Bedi and C. Sharma, "Community detection in social networks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 6, no. 3, pp. 115–135, 2016.

[78] S. X. Wu *et al.*, "Community detection in blockchain social networks," *Journal of Communications and Information Networks*, vol. 6, no. 1, pp. 59–71, 2021.

[79] W. Zhao *et al.*, "Analyzing and visualizing scientific research collaboration network with core node evaluation and community detection based on network embedding," *Pattern Recognition Letters*, vol. 144, pp. 54–60, 2021.

[80] S. Rahiminejad, M. R. Maurya, and S. Subramaniam, "Topological and functional comparison of community detection algorithms in biological networks," *BMC bioinformatics*, vol. 20, no. 1, pp. 1–25, 2019.

[81] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *Acm computing surveys (csur)*, vol. 45, no. 4, pp. 1–35, 2013.

[82] M. A. Javed *et al.*, "Community detection in networks: A multidisciplinary review," *Journal of Network and Computer Applications*, vol. 108, pp. 87–111, 2018.

[83] D. Jin *et al.*, "A survey of community detection approaches: From statistical modeling to deep learning," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[84] F. Scicchitano *et al.*, "A deep learning approach for detecting security attacks on blockchain." in *ITASEC*, 2020, pp. 212–222.

[85] S. Badillo *et al.*, "An introduction to machine learning," *Clinical pharmacology & therapeutics*, vol. 107, no. 4, pp. 871–885, 2020.

[86] M. Wang *et al.*, "Machine learning for networking: Workflow, advances and opportunities," *Ieee Network*, vol. 32, no. 2, pp. 92–99, 2017.

[87] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[88] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[89] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[90] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[91] N. Shone *et al.*, "A deep learning approach to network intrusion detection," *IEEE transactions on emerging topics in computational intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[92] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[93] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[94] L. Alzubaidi *et al.*, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[95] M. Sarhan *et al.*, "Hbfl: A hierarchical blockchain-based federated learning framework for collaborative iot intrusion detection," *Computers and Electrical Engineering*, vol. 103, p. 108379, 2022.

[96] E. T. M. Beltrán *et al.*, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.

[97] Z. Liu *et al.*, "A survey on applications of game theory in blockchain," *arXiv preprint arXiv:1902.10865*, 2019.

[98] S. Jiang and J. Wu, "Bitcoin mining with transaction fees: a game on the block size," in *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 107–115.

[99] A. Kumar and S. Jain, "Proof of game (pog): A game theory based consensus model," in *International Conference on Sustainable Communication Networks and Application*. Springer, 2019, pp. 755–764.

[100] ——, "Proof of game (PoG): a proof of work (pow)'s extended consensus algorithm for healthcare application," in *International Conference on Innovative Computing and Communications*. Springer, 2021, pp. 23–36.

[101] R. Axelrod and W. D. Hamilton, "The evolution of cooperation," *science*, vol. 211, no. 4489, pp. 1390–1396, 1981.

[102] I. Wolff, "Retaliation and the role for punishment in the evolution of cooperation," *Journal of theoretical biology*, vol. 315, pp. 128–138, 2012.

[103] N. Nikiforakis, "Punishment and counter-punishment in public good games: Can we really govern ourselves?" *Journal of Public Economics*, vol. 92, no. 1-2, pp. 91–112, 2008.

[104] L. Denant-Boemont, D. Masclet, and C. N. Noussair, "Punishment, counterpunishment and sanction enforcement in a social dilemma experiment," *Economic theory*, vol. 33, pp. 145–167, 2007.

[105] J. M. Smith, "The theory of games and the evolution of animal conflicts," *Journal of theoretical biology*, vol. 47, no. 1, pp. 209–221, 1974.

[106] D. J. Moroz *et al.*, "Double-spend counterattacks: Threat of retaliation in proof-of-work systems," *arXiv preprint arXiv:2002.10736*, 2020.

[107] J. Levin, "Wars of attrition," *Lecture Note*, 2004.

[108] J. Bulow and P. Klemperer, "The generalized war of attrition," *American Economic Review*, vol. 89, no. 1, pp. 175–189, 1999.

[109] J. M. Smith, "Evolution and the theory of games," in *Did Darwin get it right? Essays on games, sex and evolution*. Springer, 1982, pp. 202–215.

[110] J. Zhang and M. Wu, "Cooperation mechanism in blockchain by evolutionary game theory," *Complexity*, vol. 2021, no. 1, p. 1258730, 2021.

[111] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[112] Y. Chang *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[113] S. Hu *et al.*, "Large language model-powered smart contract vulnerability detection: New perspectives," in *2023 5th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 2023, pp. 297–306.

[114] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[115] Y. Zhang *et al.*, "Cumulative reasoning with large language models," *arXiv preprint arXiv:2308.04371*, 2023.

[116] M. Besta *et al.*, "Graph of thoughts: Solving elaborate problems with large language models," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17 682–17 690.

[117] S. Yao *et al.*, "Tree of thoughts: Deliberate problem solving with large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[118] J. Achiam *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[119] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[120] Y. Yao *et al.*, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[121] A. Radford *et al.*, "Improving language understanding by generative pre-training," OpenAI, Tech. Rep., 2018. [Online]. Available: https://www.openai.com/research/language-unsupervised/

[122] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[123] N. Houlsby *et al.*, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[124] L. Wang *et al.*, "Parameter-efficient fine-tuning in large language models: a survey of methodologies," *Artificial Intelligence Review*, vol. 58, no. 8, p. 227, 2025.

[125] E. J. Hu *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[126] Z. Zhang *et al.*, "Demystifying exploitable bugs in smart contracts," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 615–627.

[127] C. S. Yashavant, S. Kumar, and A. Karkare, "Scrawld: A dataset of real world ethereum smart contracts labelled with vulnerabilities," *arXiv preprint arXiv:2202.11409*, 2022.

[128] T. Durieux *et al.*, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, 2020, pp. 530–541.

[129] W. Ma *et al.*, "Combining fine-tuning and llm-based agents for intuitive smart contract auditing with justifications," *arXiv preprint arXiv:2403.16073*, 2024.

[130] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," *Cryptology ePrint Archive*, 2018.

[131] G. Xu *et al.*, "Am I eclipsed? A smart detector of eclipse attacks for Ethereum," *Computers & Security*, vol. 88, p. 101604, 2020.

[132] B. Alangot *et al.*, "Decentralized and lightweight approach to detect eclipse attacks on proof of work Blockchains," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1659–1672, 2021.

[133] Q. Dai, B. Zhang, and S. Dong, "Eclipse Attack Detection for Blockchain Network Layer Based on Deep Feature Extraction," *Wireless Communications and Mobile Computing*, vol. 2022, 2022.

[134] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*. Springer, 2017, pp. 515–532.

[135] Q. Bai *et al.*, "A deep dive into blockchain selfish mining," in *Icc 2019-2019 Ieee International Conference On Communications (Icc)*. IEEE, 2019, pp. 1–6.

[136] Z. Wang *et al.*, "Forkdec: accurate detection for selfish mining attacks," *Security and Communication Networks*, vol. 2021, pp. 1–8, 2021.

[137] M. Peterson, T. Andel, and R. Benton, "Towards detection of selfish mining using machine learning," in *International Conference on Cyber Warfare and Security*, vol. 17, no. 1, 2022, pp. 237–243.

[138] V. Chicarino *et al.*, "On the detection of selfish mining and stalker attacks in blockchain networks," *Annals of Telecommunications*, vol. 75, pp. 143–152, 2020.

[139] S.-N. Li, C. Campajola, and C. J. Tessone, "Twisted by the pools: Detection of selfish anomalies in proof-of-work mining," *arXiv preprint arXiv:2208.05748*, 2022.

[140] E. Heilman, "One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner," in *Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers 18*. Springer, 2014, pp. 161–162.

[141] R. Zhang and B. Preneel, "Publish or perish: A backward-compatible defense against selfish mining in bitcoin," in *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*. Springer, 2017, pp. 277–292.

[142] L. Luu *et al.*, "On power splitting games in distributed computation: The case of bitcoin pooled mining," in *2015 IEEE 28th Computer Security Foundations Symposium*. IEEE, 2015, pp. 397–411.

[143] Y. Zhen *et al.*, "Zero-determinant strategy for the algorithm optimize of blockchain pow consensus," in *2017 36th Chinese Control Conference (CCC)*. IEEE, 2017, pp. 1441–1446.

[144] A. Nikhalat Jahromi, A. M. Saghiri, and M. R. Meybodi, "Nik defense: An artificial intelligence based defense mechanism against selfish mining in bitcoin," *arXiv e-prints*, pp. arXiv–2301, 2023.

[145] S. A. Ghoreishi and M. R. Meybodi, "New intelligent defense systems to reduce the risks of selfish mining and double-spending attacks using learning automata," *arXiv preprint arXiv:2307.00529*, 2023.

[146] J. Lee and Y. Kim, "Preventing bitcoin selfish mining using transaction creation time," in *2018 International Conference on Software Security and Assurance (ICSSA)*. IEEE, 2018, pp. 19–24.

[147] S. Reno and S. Sultana, "Preventing selfish mining in public blockchain using alarming block and block interval time approach," in *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*. IEEE, 2022, pp. 988–993.

[148] I. Eyal, "The miner's dilemma," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 89–103.

[149] S. Elliott, "Nash equilibrium of multiple, non-uniform bitcoin block withholding attackers," in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*. IEEE, 2019, pp. 144–151.

[150] W. Sun, Z. Xu, and L. Chen, "Fairness matters: A tit-for-tat strategy against selfish mining," *Proceedings of the VLDB Endowment*, vol. 15, no. 13, pp. 4048–4061, 2022.

[151] N. Madhushanie, S. Vidanagamachchi, and N. Arachchilage, "Selfish mining attack in blockchain: a systematic literature review," *International Journal of Information Security*, vol. 23, no. 3, pp. 2333–2351, 2024.

[152] H. Kang *et al.*, "Understanding selfish mining in imperfect bitcoin and ethereum networks with extended forks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3079–3091, 2021.

[153] S. N. Mighan *et al.*, "An in-depth look at forking-based attacks in ethereum with pow consensus," *IEEE Transactions on Network and Service Management*, 2023.

[154] S. Alrubei, E. Ball, and J. Rigelsford, "Hdpoa: Honesty-based distributed proof of authority via scalable work consensus protocol for iot-blockchain applications," *Computer Networks*, vol. 217, p. 109337, 2022.

[155] S. Kokate and U. Shrawankar, "Iot data transmission security using blockchain with a trust-weighted proof of authority consensus mechanism in healthcare," *Transactions on Emerging Telecommunications Technologies*, vol. 36, no. 5, p. e70139, 2025.

[156] F. Sabrina, N. Li, and S. Sohail, "A blockchain based secure iot system using device identity management," *Sensors*, vol. 22, no. 19, p. 7535, 2022.

[157] L. Wang *et al.*, "A defense method based on deep reinforcement learning against sybil attacks in blockchain-enabled iot networks," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3702–3714, 2023.

[158] K. Venkatesan and S. B. Rahayu, "Blockchain security enhancement: an approach towards hybrid consensus algorithms and machine learning techniques," *Scientific Reports*, vol. 14, no. 1, p. 1149, 2024.

[159] F. Dewanta *et al.*, "Sibpromqtt: Protection of the mqtt communication protocol against sybil attacks applied for iot devices," in *2023 International Conference On Ic Design And Technology (Icicdt)*. IEEE, 2023, pp. 108–111.

[160] J.-P. Eisenbarth, T. Cholez, and O. Perrin, "Ethereum's peer-to-peer network monitoring and sybil attack prevention," *Journal of Network and Systems Management*, vol. 30, no. 4, p. 65, 2022.

[161] A. K. Mishra *et al.*, "Analytical model for sybil attack phases in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 379–387, 2018.

[162] W. Zhao *et al.*, "Secure blockchain-based reputation system for iiot-enabled retail industry with resistance to sybil attack," *Future Generation Computer Systems*, vol. 166, p. 107705, 2025.

[163] S. B. Prathiba *et al.*, "Fortifying federated learning in iiot: Leveraging blockchain and digital twin innovations for enhanced security and resilience," *IEEE Access*, 2024.

[164] Y. Liu *et al.*, "Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation," *arXiv preprint arXiv:2405.02580*, 2024.

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

[166] O. Zaazaa and H. El Bakkali, "Smartllmsentry: A comprehensive llm based smart contract vulnerability detection framework," *Journal of Metaverse*, vol. 4, no. 2, pp. 126–137, 2024.

[167] H. Ding *et al.*, "Smartguard: An llm-enhanced framework for smart contract vulnerability detection," *Expert Systems with Applications*, vol. 269, p. 126479, 2025.

[168] P. Ince *et al.*, "Detect llama-finding vulnerabilities in smart contracts using large language models," in *Australasian Conference on Information Security and Privacy*. Springer, 2024, pp. 424–443.

[169] L. Yu *et al.*, "Smart-llama: Two-stage post-training of large language models for smart contract vulnerability detection and explanation," *arXiv preprint arXiv:2411.06221*, 2024.

[170] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.

[171] C. T. Nguyen *et al.*, "Metachain: A novel blockchain-based framework for metaverse applications," in *2022 IEEE 95th Vehicular Technology Conference:(VTC2022-Spring)*. IEEE, 2022, pp. 1–5.

[172] Y. Fu *et al.*, "A survey of blockchain and intelligent networking for the metaverse," *IEEE Internet of Things Journal*, 2022.

[173] J. Woodward and J. Ruiz, "Analytic review of using augmented reality for situational awareness," *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[174] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[175] V. A. Traag, L. Waltman, and N. J. Van Eck, "From Louvain to Leiden: guaranteeing well-connected communities," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[176] V. D. Blondel *et al.*, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[177] E. Erdogan *et al.*, "-Zelig: Customizable Blockchain Simulator," *arXiv preprint arXiv:2107.07972*, 2021.

[178] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[179] H. Wang, Q. Yan, and V. C. Leung, "The impact of propagation delay to different selfish miners in proof-of-work blockchains," *Peer-to-Peer Networking and Applications*, pp. 1–8, 2021.

[180] A. Gervais *et al.*, "On the security and performance of proof of work blockchains," in *Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communication Security (CCS)*. ACM, 2016.

[181] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[182] J. Göbel *et al.*, "Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay," *Performance evaluation*, vol. 104, pp. 23–41, 2016.

[183] F. Erfan, M. Bellaiche, and T. Halabi, "Game-theoretic designs for blockchain-based iot: Taxonomy and research directions," in *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2022, pp. 27–37.

[184] R. Singh *et al.*, "A game theoretic analysis of resource mining in blockchain," *Cluster Computing*, vol. 23, pp. 2035–2046, 2020.

[185] T. Li *et al.*, "Semi-selfish mining based on hidden markov decision process," *International Journal of Intelligent Systems*, vol. 36, no. 7, pp. 3596–3612, 2021.

[186] ——, "Is semi-selfish mining available without being detected?" *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 10 576–10 597, 2022.

[187] M. Conti *et al.*, "A survey on security and privacy issues of bitcoin," *IEEE communications surveys & tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[188] Z. Li *et al.*, "Demystifying defi mev activities in flashbots bundle," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 165–179.

[189] Y. Kwon *et al.*, "An eye for an eye: economics of retaliation in mining pools," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 169–182.

[190] M. N. Sohail *et al.*, "Optimizing industrial iot data security through blockchain-enabled incentive-driven game theoretic approach for data sharing," *IEEE Access*, 2024.

[191] Y. Kayaba, H. Matsushima, and T. Toyama, "Accuracy and retaliation in repeated games with imperfect private monitoring: Experiments," *Games and Economic Behavior*, vol. 120, pp. 193–208, 2020.

[192] E. Budish, "The economic limits of bitcoin and the blockchain," National Bureau of Economic Research, Tech. Rep., 2018.

[193] F. Erfan, M. Bellaiche, and T. Halabi, "Efficient detection of selfish mining attacks on large-scale blockchain networks," in *2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, 2024, pp. 196–205.

[194] R. Maurya, P. Sharma, and S. Verma, "A layered architecture and middleware challenges in industrial iot: Toward resilient industry 5.0," *Journal of Industrial Information Integration*, vol. 30, p. 100456, 2025.

[195] A. Dixit, A. Trivedi, and W. W. Godfrey, "A survey of cyber attacks on blockchain based iot systems for industry 4.0," *IET Blockchain*, vol. 4, no. 4, pp. 287–301, 2024.

[196] N. Naik, "Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http," in *2017 IEEE international systems engineering symposium (ISSE)*. IEEE, 2017, pp. 1–7.

[197] Industrial Internet Consortium, "Industrial internet reference architecture," 2021, available: https://www.iiconsortium.org/IIRA.htm.

[198] F. Erfan, M. Bellaiche, and T. Halabi, "Efficient detection of selfish mining attacks on large-scale blockchain networks," in *2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 2024, pp. 196–205.

[199] F. Buccafurri, V. De Angelis, and R. Nardone, "Securing mqtt by blockchain-based otp authentication," *Sensors*, vol. 20, no. 7, p. 2002, 2020.

[200] A. Narayanan *et al.*, *Bitcoin and Cryptocurrency Technologies*. Princeton University Press, 2016.

[201] Y. Xiao *et al.*, "A survey of distributed consensus protocols for blockchain networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[202] V. Buterin, "Proof of authority chains," 2017, https://medium.com/@poanetwork/what-is-proof-of-authority-df4b469a9b4d.

[203] L. Xu and et al., "A lightweight blockchain-based architecture for industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5562–5570, 2021.

[204] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 281–310, 2015.

[205] H. Yu *et al.*, "A survey on consortium blockchain consensus mechanisms," *Journal of Computer Science and Technology*, vol. 35, pp. 913–930, 2020.

[206] Ethereum Foundation, "Ethereum node records (enr)," https://eips.ethereum.org/EIPS/eip-868, 2019, ethereum Improvement Proposal 868.

[207] ——, "Discovery v5 topic advertisement," https://eips.ethereum.org/EIPS/eip-1459, 2019, ethereum Improvement Proposal 1459.

[208] Protocol Labs, "libp2p project documentation, protocol overview," https://docs.libp2p.io/, 2021.

[209] J. Mao *et al.*, "Sybilhunter: Hybrid graph-based sybil detection by aggregating user behaviors," *Neurocomputing*, vol. 500, pp. 295–306, 2022.

[210] P. Gao *et al.*, "Sybilfuse: Combining local attributes with global structure to perform robust sybil detection," in *2018 IEEE conference on communications and network security (CNS)*. IEEE, 2018, pp. 1–9.

[211] Q. Stokkink, S. Roos, and J. Pouwelse, "Web3 sybil attack mitigation: Current solutions and the road ahead," in *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2023, pp. 57–66.

[212] H. Lu *et al.*, "Sybilhp: Sybil detection in directed social networks with adaptive homophily prediction," *Applied Sciences*, vol. 13, no. 9, p. 5341, 2023.

[213] H. Kavak *et al.*, "Simulation for cybersecurity: state of the art and future directions," *Journal of Cybersecurity*, vol. 7, no. 1, p. tyab005, 2021.

[214] A. Shikfa, M. Önen, and R. Molva, "Privacy-preserving sybil defense in p2p systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 729–742, 2011.

[215] S. Buchegger and J.-Y. Le Boudec, "Reputation systems for self-organized networks: Issues and challenges," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2005, pp. 1–14.

[216] S. Saxena and V. Sejwar, "Sybil attack detection and analysis of energy consumption in cluster based sensor networks," *International Journal of Grid and Distributed Computing*, vol. 7, no. 5, pp. 15–30, 2014.

[217] H. Wang, L. Ma, and H.-y. Bai, "A three-tier scheme for sybil attack detection in wireless sensor networks," in *2020 5th international conference on computer and communication systems (ICCCS)*. IEEE, 2020, pp. 752–756.

[218] G. Wang *et al.*, "Neighbor similarity trust against sybil attack in p2p e-commerce," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 3, pp. 824–833, 2014.

[219] Y. Zhu *et al.*, "Sybil attacks detection and traceability mechanism based on beacon packets in connected automobile vehicles," *Sensors*, vol. 24, no. 7, p. 2153, 2024.

[220] C. Zhang *et al.*, "A survey on deep learning-based anomaly detection in industrial time series," *Neurocomputing*, vol. 411, pp. 92–107, 2021.

[221] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[222] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[223] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[224] A. G. Roy *et al.*, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[225] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[226] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: foundations, design landscape and research directions," *arXiv preprint arXiv:1608.00771*, 2016.

[227] I. Lütkebohle, "BWorld Robot Control Software," https://dasp.co/, 2008, [Online; accessed 19-July-2008].

[228] C. F. Torres, M. Steichen *et al.*, "The art of the scam: Demystifying honeypots in ethereum smart contracts," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1591–1607.

[229] DASP: The DApp Security Project, "Dasp top 10: Top 10 smart contract vulnerabilities," https://dasp.co/, 2020, accessed: 2025-06-06.

[230] J. Gao *et al.*, "Easyflow: Keep ethereum away from overflow," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*. IEEE, 2019, pp. 51–54. [Online]. Available: https://dl.acm.org/doi/10.1109/ICSE-Companion.2019.00029

[231] J. Zhang *et al.*, "Ethereum smart contracts: Vulnerabilities and security tools," *IEEE Access*, vol. 8, pp. 6463–6476, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8962943

[232] Z. Liu *et al.*, "Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion," in *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*. IJCAI, 2021, pp. 2792–2798. [Online]. Available: https://www.ijcai.org/proceedings/2021/0379.pdf

[233] M. AI, "Meta llama 3.1 8b-instruct," https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct, 2024, accessed: 2025-05-30.

[234] Y. Zheng *et al.*, "Llamafactory: Unified efficient fine-tuning of 100+ language models," *arXiv preprint arXiv:2403.13372*, 2024.

[235] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[236] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," https://github.com/UKPLab/sentence-transformers, 2019, accessed: 2025-05-28.

[237] W. Wang *et al.*, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in neural information processing systems*, vol. 33, pp. 5776–5788, 2020.

[238] OpenAI, "Gpt-4.1 mini model documentation," https://platform.openai.com/docs/models/gpt-4.1-mini, 2024, accessed: 2025-05-30.

[239] N. K. Tran, M. A. Babar, and J. Boan, "Integrating blockchain and internet of things systems: A systematic review on objectives and designs," *Journal of Network and Computer Applications*, vol. 173, p. 102844, 2021.

[240] J. Sengupta, S. Ruj, and S. D. Bit, "A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT," *Journal of Network and Computer Applications*, vol. 149, p. 102481, 2020.

[241] M. Saad *et al.*, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1977–2008, 2020.

[242] S. Roy *et al.*, "A survey of game theory as applied to network security," in *2010 43rd Hawaii International Conference on System Sciences*. IEEE, 2010, pp. 1–10.

[243] S. Xia *et al.*, "A bayesian game based vehicle-to-vehicle electricity trading scheme for blockchain-enabled internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 6856–6868, 2020.

[244] J. Qiu *et al.*, "Blockchain-based secure spectrum trading for unmanned-aerial-vehicle-assisted cellular networks: An operator's perspective," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 451–466, 2019.

[245] A. Wilczyński and J. Kołodziej, "Modelling and simulation of security-aware task scheduling in cloud computing based on blockchain technology," *Simulation Modelling Practice and Theory*, vol. 99, p. 102038, 2020.

[246] Y. Liu *et al.*, "Decentralized resource allocation for video transcoding and delivery in blockchain-based system with mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 169–11 185, 2019.

[247] Z. Xiong *et al.*, "Optimal pricing-based edge computing resource management in mobile blockchain," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[248] Z. Liu *et al.*, "A survey on blockchain: A game theoretical perspective," *IEEE Access*, vol. 7, pp. 47 615–47 643, 2019.

[249] S.-K. Kim, "Blockchain governance game," *Computers & Industrial Engineering*, vol. 136, pp. 373–380, 2019.

[250] H. Wang *et al.*, "Blockchain-based resource allocation model in fog computing," *Applied Sciences*, vol. 9, no. 24, p. 5538, 2019.

[251] R. Kovacs *et al.*, "A collaborative game theory approach for determining the feasibility of a shared as blockchain infrastructure," in *2021 20th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2021, pp. 1–6.

[252] J. Cheng *et al.*, "A survey of security threats and defense on blockchain," *Multimedia Tools and Applications*, vol. 80, no. 20, pp. 30 623–30 652, 2021.

[253] D. Xu *et al.*, "Game theoretic study on blockchain based secure edge networks," in *2017 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2017, pp. 1–5.

[254] K. Toda, N. Kuze, and T. Ushio, "Game-theoretic approach to a decision-making problem for blockchain mining," *IEEE Control Systems Letters*, vol. 5, no. 5, pp. 1783–1788, 2020.

[255] B. Johnson *et al.*, "Game-theoretic analysis of DDoS attacks against bitcoin mining pools," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 72–86.

[256] S. Dey, "Securing majority-attack in blockchain using machine learning and algorithmic game theory: A proof of work," in *2018 10th computer science and electronic engineering (CEEC)*. IEEE, 2018, pp. 7–10.

[257] H. Tan and I. Chung, "Secure authentication and key management with blockchain in vanets," *IEEE access*, vol. 8, pp. 2482–2498, 2019.

[258] L. Cheng *et al.*, "SCTSC: A semicentralized traffic signal control mode with attribute-based blockchain in iovs," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 6, pp. 1373–1385, 2019.

[259] A. Taghizadeh, H. Kebriaei, and D. Niyato, "Mean field game for equilibrium analysis of mining computational power in blockchains," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7625–7635, 2020.

[260] X. Chen and X. Zhang, "Secure electricity trading and incentive contract model for electric vehicle based on energy blockchain," *IEEE access*, vol. 7, pp. 178 763–178 778, 2019.

[261] V. Hassija *et al.*, "Dagiov: A framework for vehicle to vehicle communication using directed acyclic graph and game theory," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4182–4191, 2020.

[262] L. W. Cong, Z. He, and J. Li, "Decentralized mining in centralized pools," *The Review of Financial Studies*, vol. 34, no. 3, pp. 1191–1235, 2021.

[263] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8076–8094, 2019.

[264] Y. Zuo, S. Jin, and S. Zhang, "Computation offloading in untrusted MEC-aided mobile blockchain IoT systems," *IEEE Transactions on Wireless Communications*, 2021.

[265] Z. Xiong *et al.*, "Cloud/fog computing resource management and pricing for blockchain networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4585–4600, 2018.

[266] S. Guo *et al.*, "Blockchain meets edge computing: Stackelberg game and double auction based task offloading for mobile blockchain," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5549–5561, 2020.

[267] X. Ding *et al.*, "An incentive mechanism for building a secure blockchain-based internet of things," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 477–487, 2020.

[268] Z. Xiong *et al.*, "When mobile blockchain meets edge computing," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 33–39, 2018.

[269] W. Liu *et al.*, "A distributed game theoretic approach for blockchain-based offloading strategy," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[270] W. Guo *et al.*, "Resource allocation for edge computing-based blockchain: A game theoretic approach," in *2020 IEEE international conference on communications workshops (ICC workshops)*. IEEE, 2020, pp. 1–6.

[271] J. Yuan *et al.*, "Edge mining resources allocation among normal and gap blockchains using game theory," *The Journal of Supercomputing*, pp. 1–18, 2022.

[272] X. Ding *et al.*, "Pricing and budget allocation for IoT blockchain with edge computing," *IEEE Transactions on Cloud Computing*, 2022.

[273] X. Yang *et al.*, "Automated demand response framework in elns: Decentralized scheduling and smart contract," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 50, no. 1, pp. 58–72, 2019.

[274] H. Yao *et al.*, "Resource trading in blockchain-based industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3602–3609, 2019.

[275] A. S. Bataineh *et al.*, "A game-based secure trading of big data and iot services: Blockchain as a two-sided market," in *International Conference on Service-Oriented Computing*. Springer, 2020, pp. 85–100.

[276] H. Chen *et al.*, "Smartstore: A blockchain and clustering based intelligent edge storage system with fairness and resilience," *International Journal of Intelligent Systems*, vol. 36, no. 9, pp. 5184–5209, 2021.

[277] J. Gao *et al.*, "Supply chain equilibrium on a game theory-incentivized blockchain network," *Journal of Industrial Information Integration*, p. 100288, 2021.

[278] B. Zhang *et al.*, "A contract game for direct energy trading in smart grid," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2873–2884, 2016.

[279] J. Li *et al.*, "Decentralized on-demand energy supply for blockchain in internet of things: A microgrids approach," *IEEE transactions on computational social systems*, vol. 6, no. 6, pp. 1395–1406, 2019.

[280] Z. Li *et al.*, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE transactions on industrial informatics*, vol. 14, no. 8, pp. 3690–3700, 2017.

[281] M. Ali *et al.*, "Efficient and secure energy trading in internet of electric vehicles using IOTA blockchain," in *2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET)*. IEEE, 2020, pp. 87–91.

[282] X. Fu, H. Wang, and P. Shi, "A survey of blockchain consensus algorithms: mechanism, design and applications," *Science China Information Sciences*, vol. 64, no. 2, pp. 1–15, 2021.

[283] N. Alzahrani and N. Bulusu, "Towards true decentralization: A blockchain consensus protocol based on game theory and randomness," in *International conference on decision and game theory for security*. Springer, 2018, pp. 465–485.

[284] Y. Wei *et al.*, "A modified blockchain dpos consensus algorithm based on anomaly detection and reward-punishment," in *2021 13th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2021, pp. 283–288.

[285] L. Di *et al.*, "The existence of consensus equilibria for data trading under the framework of internet of things (IoT) with blockchain ecosystems," *Procedia Computer Science*, vol. 174, pp. 55–65, 2020.

[286] B. N. Alhasnawi *et al.*, "Consensus algorithm-based coalition game theory for demand management scheme in smart microgrid," *Sustainable Cities and Society*, vol. 74, p. 103248, 2021.

[287] Y. Jiang, Y. Zhong, and X. Ge, "IIoT data sharing based on blockchain: a multi-leader multi-follower stackelberg game approach," *IEEE Internet of Things Journal*, 2021.

[288] C. Zhang, T. Shen, and F. Bai, "Toward secure data sharing for the iot devices with limited resources: A smart contract–based quality-driven incentive mechanism," *IEEE Internet of Things Journal*, 2022.

[289] F. Bai *et al.*, "Trustworthy blockchain-empowered collaborative edge computing-as-a-service scheduling and data sharing in the IIoE," *IEEE Internet of Things Journal*, 2021.

[290] H. Si *et al.*, "Iot information sharing security mechanism based on blockchain technology," *Future generation computer systems*, vol. 101, pp. 1028–1040, 2019.

[291] R. Akkaoui, X. Hei, and W. Cheng, "An evolutionary game-theoretic trust study of a blockchain-based personal health data sharing framework," in *2020 Information Communication Technologies Conference (ICTC)*. IEEE, 2020, pp. 277–281.

[292] S. Jiang and J. Wu, "Game theoretic storage outsourcing in the mobile blockchain mining network," in *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2020, pp. 300–308.

[293] Y. Wang *et al.*, "Pool strategies selection in pow-based blockchain networks: Game-theoretic analysis," *IEEE Access*, vol. 7, pp. 8427–8436, 2019.

[294] S. Feng *et al.*, "On cyber risk management of blockchain networks: A game theoretic approach," *IEEE Transactions on Services Computing*, vol. 14, no. 5, pp. 1492–1504, 2018.

[295] J.-M. Lasry and P.-L. Lions, "Mean field games," *Japanese journal of mathematics*, vol. 2, no. 1, pp. 229–260, 2007.

APPENDIX A
GAME-THEORETIC DESIGNS FOR BLOCKCHAIN-BASED IOT: TAXONOMY AND RESEARCH DIRECTIONS

Fatemeh Erfan, Martine Bellaiche, Talal Halabi

Published on IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Newark, CA, USA, pp. 27–37, August 15, 2022

Abstract

In the last few years, the combination of the Internet of Things (IoT) and blockchain technologies has widely attracted significant research attention from academia and the industry due to the salient characteristics of blockchain and its natural compatibility with the functional requirements of IoT architectures. Despite the advantages of utilizing the blockchain in IoT systems, several issues have been raised regarding the mitigation of security attacks on blockchain-based IoT systems, management of blockchain operations, node communications, energy consumption, efficient consensus algorithm design, data sharing, and edge/fog computing. Many of the solutions proposed in the literature tried to address these issues using game-theoretic models. This paper surveys and investigates state-of-the-art blockchain-based IoT solutions leveraging game theory to address common issues in blockchain design. The developed taxonomy enabled us to highlight existing challenges to the integration of game theory models into the design of blockchain-driven IoT and bring forward some critical research questions.

Keywords: Internet of Things, blockchain design, game theory, security, consensus algorithms

Introduction

The security and privacy of the Internet of Things (IoT) have attracted tremendous research efforts. IoT devices generate and process data that may contain sensitive information, creating vulnerabilities that can be exploited by confidentiality and integrity threats. To address these, traditional cryptographic techniques such as attribute-based encryption, identity-based encryption, and public-key systems have been applied. However, they tend to be expensive in terms of energy consumption, processing, and storage. In addition, they require highly centralized architectures, which raise several deployment and scalability challenges and exacerbate single points of failure. Blockchain technology provides a platform to tackle IoT

security concerns and alleviate the inherent bottleneck of fast-growing IoT networks that rely on centralized servers [25].

Despite the significant benefits of integrating the blockchain within IoT, it entails several challenges that can be broken down into seven major areas: security, blockchain operations, communication, edge/fog computing, energy consumption, consensus algorithm, and data sharing in scenarios involving devices with constrained capabilities. Furthermore, emerging blockchain-based IoT systems should be analyzed and compared against state-of-the-art architectures in terms of security, privacy, communication, and storage overheads. Moreover, despite the benefits of blockchain in mitigating some security attacks against IoT, including single point of failure, Sybil, flooding, and jamming [21], there exist other security attacks driven by the blockchain itself that could be launched against the IoT platform. These include selfish, withholding, majority, and Distributed Denial of Service (DDoS) attacks [97]. To avoid these threats, some researchers usually investigate and analyze the strategies executed by the IoT nodes using game theory models that can be effectively applied to predict the behavior of participating nodes and their defense strategies. Also, during the consensus algorithm, semi-honest or malicious entities participating in the system intend to maximize their benefits by deviating from the protocol. Hence, these nodes will not broadcast their mined blocks; instead, they will hold on to them to maximize their revenue. In such scenarios, game-theoretic approaches can be applied to enable the system to strategically analyze the communications among participants to predict malicious behaviors and adopt optimized punishment strategies toward selfish nodes.

In general, game theory can be leveraged in blockchain-based IoT systems to address existing challenges. For instance, by adopting the blockchain, the design must determine how to manage computational resources, maximize rewards, and choose the nonce selection strategy. In particular, in a blockchain like Bitcoin, participants exploit their computational resources to solve the difficult cryptographic puzzle and gain a reward. In blockchain-based IoT, devices joining the system normally have limited computational resources and must compete with each other to gain access to limited energy resources to support computing. Hence, game theory models can be used to solve such non-cooperative optimization problems. Last, but not least, game theory can be used in designing the consensus algorithm. As the miners in the blockchain are IoT devices with limited computational resources, performing Proof of Work (PoW) or other consensus algorithms may not be suitable. Therefore, Proof of Game (PoG) has been proposed to improve the efficiency of IoT systems [32, 98–100].

The goal of this paper is to attract the attention of the research community toward the potential applications of game theory in blockchain-based IoT systems. To the best of our knowledge, this is the first work focusing on the applications and benefits of game theory in

blockchain-based IoT systems. The contributions of this paper are summarized as follows:

- We overview the various game theory models that have been integrated into the design of blockchain-based IoT.
- We survey, analyze, and classify existing work presenting game-theoretic approaches to tackle security, performance, and management issues in blockchain-based IoT systems into a comprehensive and timely taxonomy.
- We provide an insightful discussion and highlight the research questions and open challenges related to exploiting game theory for blockchain designs in future IoT.

The remainder of the paper is organized as follows. Section A presents background information and concepts. Section A proposes a taxonomy of game theory applied to blockchain-based IoT. Section A discusses the research gaps and future research areas. Finally, Section A concludes the paper.

Background Concepts

Blockchain-based IoT systems normally contain the blockchain as a component within their global architecture [239]. This section provides background knowledge of IoT and blockchain as well as game-theoretic concepts to allow the reader to acquire a comprehensive understanding of Blockchain-based IoT and better grasp its design challenges. Blockchain is one of the disruptive technologies of the last decade. It provides a distributed platform to run decentralized service architectures such as IoT and edge computing [21], and can be effectively used to respond to the security and privacy requirements of most IoT domains by enabling a secure and transparent ledger system [22]. The literature provides several surveys on security issues and blockchain solutions in IoT systems [25, 38, 240]. Nonetheless, the application of game theory to address blockchain design challenges in various scenarios has also been gaining significant traction [97].

IoT Systems

IoT has brought new service paradigms across different sectors such as health, autonomous transportation, and various industries. It plays a crucial role in turning homes into smart homes and cities into smart cities [22]. IoT involves smart devices that can upload data to the Internet and control the decisions of cyber-physical processes. They can be monitored and controlled remotely using mobile applications such as in smart hospitals and factories.

IoT consists of four main layers: the perception layer, consisting of sensors and actuators; the network layer (edge-fog-cloud communications) responsible for transferring the data from devices for processing in upper layers; the processing layer, leveraging the cloud environment to perform some computational tasks; and the application layer delivered via end-user devices. In all layers, data confidentiality, integrity, and privacy need to be maintained.
IoT raises many vulnerabilities that can lead to security threats. Attacks at the perception layer include side channel, device tampering, and fake node injection [240]. Attacks at the networking layer include Sybil and man-in-the-middle. At the processing layer, data corruption and DDoS attacks could affect the cloud server and delivered services. Finally, attacks carried out against the application layer are mainly driven by malware such as viruses, worms, rootkits, and ransomware.
Several security attacks and privacy issues in IoT can be addressed using the blockchain [21]. These include the lack of trusted communication channels (intra-system and inter-system), threats brought by single points of failure, and data integrity (e.g., can be addressed using consensus algorithms). Blockchain provides an infrastructure to implement secure and efficient IoT systems by effectively integrating its secure and decentralized platform, and can be applied in most IoT domains such as the Internet of Vehicles (IoV), autonomous drones, smart grids, healthcare IoT, and supply chains.

Blockchain Technology

The blockchain is a decentralized system that allows transactions to be verified by a group of unreliable entities [20]. For instance, Bitcoin is a Blockchain concept and one of the virtual cryptocurrencies that has revolutionized the mechanisms of money transfer. To best understand emerging blockchain-based IoT systems, it is important to acquire basic knowledge of the components, types, and significant features of blockchain.
Block: Each transaction transmitted through the blockchain is stored in a block. A sequential chain of blocks created by miners is called a blockchain. Each block includes two parts, the header and body. Data as a transaction is stored in the body part of a data structure called the Merkle tree. It keeps hashing transactions in pairs until a single hash called the Merkle Root is obtained [22]. The block header consists of the identity of the current block, the previous block, a timestamp, and a nonce.
Miner: Once a new block is created by a miner, it is added to the blockchain through the following steps. Miners perform a consensus algorithm to solve a cryptographic puzzle. Then, members in the blockchain validate the created block. To verify a transaction, a local copy of all transactions is not required, and the verification phase can be done by using the Merkle tree stored in the block. Finally, a reward is given to the miner, and verified transactions are

stored in the blockchain.
Consensus algorithm: In general, it is used to validate the blockchain and add new blocks. This is an approach to mining a block through the blockchain. As part of all consensus algorithms, participants should verify the transaction stored in a new block. Every transaction has a unique ID, which is the cryptographic hash of the corresponding transaction's information stored in the block. Table A.1 compares the consensus protocols used in different blockchain platforms [26, 30–32].

- *Proof of Work (PoW)* is the first consensus algorithm used in blockchain, such as the Bitcoin network. It requires the participants to use their capacities for solving the hard cryptographic puzzle. Publishing new blocks under the PoW protocol is called "mining". Miners try to find a nonce and should have powerful computational resources to perform the proof of work.

- *Proof of Stake (PoS)* is used in blockchains such as Ethereum, where the term "miner" is replaced with "validator". Here, there is no race among participants to find the nonce value and solve the puzzle. One validator is selected for mining based on its proportional stake, including its economic share in the network. This is done in a pseudorandom way with the probability of selection proportional to the validators' share [25–27]. Delegated Proof of Stake (DPoS) is a variant of PoS where all peers vote to select a node as a witness or delegate [26]. Witnesses are rewarded for generating new blocks, and delegates oversee retaining the blockchain and offering some changes, including transaction fees and block size. DPoS is more efficient than PoS thanks to its different voting and delegation mechanism.

- *Proof of Game (PoG)* is one of the consensus protocols leveraging game-theoretic models. PoG is proposed for resource-dependent and computational independent consensus algorithms. It restricts the block creator in creating the number of blocks based on the level of the game it can play and the score it can earn [99].

- *Practical Byzantine Fault Tolerance (PBFT)* is a popular consensus algorithm used in permissioned blockchain, where the system combines a group of active and passive replicas [241]. The pre-preparation phase is the first stage of the process of deploying PBFT, followed by the preparation, commit, and reply stages.

- *Proof of Activity (PoA)* combines the features of PoW and PoS. It tends to maintain the best aspects of both algorithms and avoid their worst features. The process of mining is similar to that of PoW. In the first stage, miners make efforts to solve the

hash function and win the race based on PoS. Next, transactions are added to the block. Then, several validators are selected to verify the new block according to the PoS consensus algorithm.

Table A.1 categorizes different consensus algorithms which are suitable for IoT networks based on accessibility, scalability, computing, network, and storage overhead, throughput, and latency. PBFT has low computing overhead while PoW, PoG, and PoA have high computing overhead.
Based on how it is used and deployed in various systems, blockchain can be categorized into the following types:

- *Public blockchain*, where everyone can participate in the mining, publishing, and verification of the new block as well as accessing the data gathered in the blockchain. This blockchain is also known as permissionless, i.e., everyone can join the blockchain without any permission.
- *Private blockchain*, where only known participants can join the blockchain and access the data shared in the ledger. This type is also named permissioned blockchain.
- *Consortium*, also known as a federated blockchain, this type of ledger is similar to a private blockchain as it includes clusters of private participants managed by different organizations. Hence, multiple organizations with private participants usually join a consortium blockchain.

Blockchain technology also enjoys several features that make it attractive for IoT research, including decentralization, immutability, auditability, and fault tolerance [25]. Decentralization is one of the most salient characteristics of blockchain. Turning from a centralized towards a decentralized architecture will protect the IoT network against single points of failure and bottlenecks [36] and enable every device to participate in the consensus protocol. Each transaction in the blockchain should be verified by the majority of participants and no central entity monitors the information generated through the network.

Game Theory

Game theory can play a key role in addressing strategic situations in IoT networks, such as competition over resources and decentralized decision making. Also, researchers can take advantage of the powerful game-theoretic models to formally assess the security issues in blockchain-based IoT and develop a strategic defense against attacks [22]. In peer-to-peer

Table A.1 Comparison of consensus algorithms.

Consensus algorithm	Accessibility	Scalability	Computing overhead	Network overhead	Storage overhead	Throughput	Latency
PoW	Public	High	High	Low	High	Low	High
PoS	Public	High	Medium	Low	High	Low	Medium
DPoS	Public	High	Medium	N/A	High	High	Medium
PBFT	Private	Low	Low	High	High	High	Low
PoG	N/A	Low	High	Medium	Low	N/A	N/A
PoA	Public	High	High	Low	High	Low	Medium

systems, game theory provides a set of mathematical tools to model the communications between peers, predict their behaviors, and optimize their actions. Generally, each player participating in the game aims to choose a strategy that maximizes their reward. This section overviews the game-theoretical models that have been incorporated into blockchain-based IoT systems.

To better grasp the power and advantages of game theory, some basic terms are first explained as follows.

- *Player:* Each entity (e.g., individual or organization) participating in the game and affecting it by their moves.
- *Action:* In every game, there is a finite set of actions that players can take as part of their strategies.
- *Payoff:* Players take actions to maximize the revenue or payoff they receive. The payoff is defined based on players' selected strategies and is computed using problem-specific parameters.
- *Strategy:* Players' reward or payoff is computed according to the strategies played by all players. Players can choose between a pure or mixed (probabilistic) strategy over their set of actions.
- *Solution concept*: It is a set of rules used to predict how a game will be played. The predictions derived by solving the game indicate the strategies that players will select.
- *Nash equilibrium:* It is a solution concept. If each player has chosen a strategy and no player has an incentive to maximize their own expected payoff by deviating from their strategy while the other players keep theirs unchanged, the game is in a Nash equilibrium. When the Nash equilibrium is achieved, each player has chosen their optimal strategy that maximizes their reward.

Many game-theoretical models have been proposed to design blockchain networks, especially in IoT systems. These are divided between non-cooperative and cooperative (i.e., coalitional)

games. Due to their inherent nature, non-cooperative games are the most used in blockchain network design, normally to optimize the defensive strategy (e.g., honest mining) against the adversary's actions. On the other hand, cooperative games are used to effectively coordinate the collaborative behavior of nodes in peer-to-peer networks. As shown in Figure A.1, non-cooperative games can either be static or dynamic, and can be further categorized according to the information available about players' types and their action history [242]. For instance, acquiring the full information in the decentralized blockchain network in a timely fashion can be challenging, hence Bayesian games can be adopted in the design of distributed blockchain-enabled IoT systems with incomplete information sharing [243].
Various game models that exist under this classification have been leveraged for blockchain-based IoT design. These are described as follows:

- *Stackelberg game:* An asymmetric, sequential game where players are divided into two types: the leader who makes a move first, and the followers who select their strategies after observing the leader's actions. Every player tries to maximize their payoff by selecting the best strategy. The leader derives the optimal strategy through backward-inductive reasoning about the follower's anticipated actions. Many of the approaches discussed in this paper used the Stackelberg game model [244–247].

- *Repeated game:* When players dynamically play a game multiple times (e.g., super game), the previous strategies may be observed by each player so that they can optimize their future strategies accordingly. Hence, dynamic games can be based on complete or incomplete information.

- *Stochastic game:* It involves repeating distinct games every time the game is executed, i.e., the game is played in different states. Players are allowed to rearrange their strategies depending on the decisions made by other players [248]. For instance, miners can leverage this game to find the best chain to join and the best block to be mined

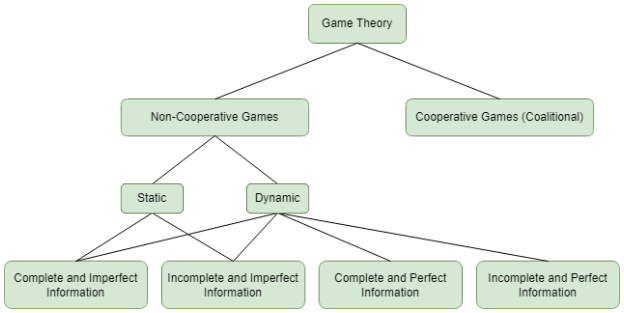


Figure A.1 Categorization of game theory models.

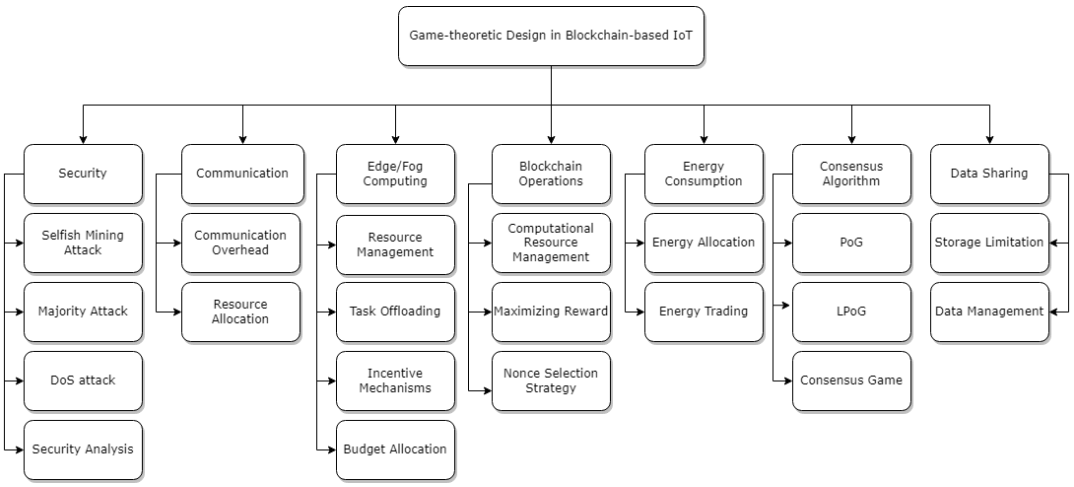


Figure A.2 A timely and comprehensive taxonomy of game-theoretic design in blockchain-based IoT systems.

and verified. A stochastic game has also been used to mitigate the majority attack, as described later [249].

- *Differential game*: Here, each player solves an optimal control problem. For instance, Wang et al. [250] used differential games to model the blockchain on top of fog computing to solve the resource contribution problem.

- *Coalitional game:* An important approach to studying the interactions of cooperative players. Each formed coalition has a payoff that may be distributed among its members (e.g., transferable utility). For instance, this approach has been used to investigate the feasibility of integrating blockchain into the design of autonomous systems [251].

For distributed (and mostly non-collaborative) decision making in IoT networks, game theory can be a valuable tool to predict and regulate nodes' behaviors in the blockchain. Also, it can be used for detecting and mitigating security attacks.

Taxonomy of Game-theoretic Design in Blockchain-based IoT

Blockchain is a potential solution to tackle security issues in emerging IoT ecosystems. Nonetheless, it entails challenges in terms of performance, management, and energy consumption, most of which have been recently addressed by integrating game-theoretic models. This section builds a comprehensive taxonomy of the integration of game-theoretic models into various design areas in blockchain-based IoT, including security, communication, blockchain operations, energy consumption, consensus protocols, and data sharing, as illus-

trated in Figure A.2. This taxonomy is a valuable roadmap for conducting future research in the field of blockchain design for IoT.

Game Theory for Security

When deploying blockchain-based IoT systems, there exist serious security threats that should not be overlooked [252], especially zero-day attacks. For instance, Dongjin Xu et. al. [253] proposed a game-theoretic punishment approach based on action records in the blockchain platform to decrease the probability of occurrence of cyber attacks by mobile users and edge servers and improve the security of the edge network. This section describes several important attacks along with the game-theoretic approaches proposed to address them.

Selfish mining attack

A subversive strategy in PoW-driven blockchain where malicious miners or mining pools do not broadcast the newly mined block but instead select to withhold or hold on to it, then release it later [54]. In this case, honest miners waste their computational power, creating a block race between honest and selfish miners. Selfish mining attacks can invalidate the blocks mined by honest miners. Thus, all transactions in the honest miners' blocks get rejected. One of the common selfish mining attacks is the pool block withholding (PBWH) attack, where a malicious pool launches a Block Withholding (BWH) attack against another pool. To prevent this, it is important to assess the strategies taken by miners during communication. Several game-theoretic approaches have been proposed to optimally solve the decision-making problem of resource mining in blockchain [184, 254].
For instance, Singh et al. [184] proposed a non-cooperative differential game to maximize the profit of miners, where each miner behaves selfishly and individually wants to maximize the profit gained. On the other hand, Toda et al. [254] modeled the decision-making problem of miners as a non-cooperative game, where the utility function is defined in terms of energy consumption and the mean mining reward.
For example, Eyal [148] proposed a non-cooperative game among the pools to analyze the node communication through the blockchain. Particularly, the miners are willing to join those private pools that are not under the PWH attack. Thus, large mining pools could be shrunken down into several smaller ones. The proposed miner's dilemma is similar to the classical prisoner's dilemma: If a miner controls the majority of the mining power, both miners will achieve a lesser reward than when both miners behave honestly. Hence, each miner can choose to attack or behave honestly. If miner 2 (miner 1) chooses to behave maliciously or honestly, the payoff of miner 1 (miner 2) is larger when behaving maliciously than when

behaving honestly, respectively. At the equilibrium state, if both miners behave maliciously, the payoff of each miner is smaller than its payoff if neither miner behaves maliciously. The game is played dynamically. The miners can cooperate to behave honestly, and in each round, a miner can detect whether another miner is behaving maliciously and deviating from the agreement. TABLE A.2 depicts this interaction.

Majority attack

When a single miner, a group of Sybil nodes, or a mining pool controls more than 50% of the computational power of a network, a majority or 51% attack [54] is yielded. Attackers can prevent blocks from being verified and reverse transactions during the time they are in control to allow double-spending. Moreover, malicious nodes can split the blockchain and fork the main network. They can also prevent honest miners from finding and verifying new blocks [241]. To avoid this, benign miners can defend against attacker nodes by adding honest nodes to the blockchain network [249]. The communication between these nodes can also be formulated as a stochastic game in which the states reflect the blockchain portions controlled by malicious and benign nodes.

DoS attack

Mining pools can potentially carry out DoS attacks to maximize their reward. Some may launch DDoS attacks to decrease the other mining pools' rewards or to access additional computation resources and improve the likelihood of solving the next proof of work. Non-cooperative games can be used to prevent DoS attacks. Particularly, Johnson et al. [255] addressed the trade-off between two different strategies when performing the consensus protocol, including construction and destruction by deploying a series of game-theoretic models. Under the construction strategy, a mining pool uses additional computing resources to increase the probability of winning the next race. Under the destruction strategy, it launches a computationally intensive DDoS attack to mitigate the likelihood of winning for the other pools. The authors also studied the incentives of Bitcoin mining pool operators to launch DDoS attacks. Furthermore, machine learning for anomaly detection combined with game

Table A.2 The miner's Dilemma (simplified), where p_1 and p_2 represent the payoffs of miners 1 and 2, respectively.

Miner 1 / Miner 2	Honest	Malicious
Honest	$(p_1 = 1, p_2 = 1)$	$(p_1 > 1, p_2 < 1)$
Malicious	$(p_1 < 1, p_2 > 1)$	$(p_1 < 1, p_2 < 1)$

theory can be helpful in identifying various attacks, including majority and DDoS [256].

Security and performance analysis

Game theory is one of the tools used to analyze and compare the security of blockchain-based IoT systems and evaluate their performance. For instance, Tan and Chung [257] made use of game theory to evaluate the security of their proposed blockchain-enabled VANETs system against not only unauthorized tracking of specified vehicles but also adaptive chosen message and replay attacks. Cheng et al. [258] utilized a game theory-based method to analyze time cost consideration in the traffic signal control mode in blockchain-based IoV.
On the other hand, mean-field game is a popular game-theoretic model dealing with large-scale settings and was used by Taghizadeh et al. [259] for equilibrium analysis of mining computational power in blockchain-based IoT. Since dealing with a large group of peers in the blockchain is one of the major challenges that conventional analytical tools normally face, leveraging mean-field game theory allows to analyze the behavior of a large number of players in the network by enabling macroscopic security studies of mining groups.

Game Theory for Communications

Communication problems in Blockchain-enabled IoT networks involve the minimization of communication overhead (i.e., transmitted packets) and the management of communication resources (e.g., how to allocate radio channels). Game theory has been applied to solve these problems. For instance, Qiu et al. [244] used a Stackelberg game to perform spectrum trading operations among the buyer and seller. The game involves a leader (mobile network operator) and multiple trackers (unmanned aerial vehicles) competing over resources.
One of the most crucial challenges that have emerged in electric vehicles (EV) is to ascertain the best strategy for each decision-maker to optimize the interests of any participant in the grid with a lower communication overhead. Generally, game theory can be utilized when multiple parties are willing to maximize their own revenue based on their abilities and interests. Xia et al. [243] proposed an approach using a Bayesian game to achieve optimal pricing in the distributed blockchain-based IoV while decreasing the communication load. More specifically, an electricity buyer buys electricity from an electricity seller, which is a vehicle that wants to sell its electricity in a trustworthy way by using a blockchain platform. The Bayesian game-based pricing scheme is written by the smart contract, which plays the role of a virtual agency and achieves optimal trading with lower communication costs.
Chen et al. [260] proposed a secure electricity trading and incentive contract model using blockchain and game theory to encourage vehicles to trade and participate in the game,

which is based on income and reward. They showed that communication overhead could be reduced by 64.55%. Hassija et al. [261] proposed an IoV framework using game theory to manage the communication among vehicles. They used the blockchain to address the security and tracking challenges via game-theoretic modeling of the interactions between the vehicles providing and consuming offloading services. Furthermore, to maximize miners' utility, Cong et al. [262] enabled the miners to form a coalition (e.g., mining pool) and cooperate by following the reward allocation mechanism. Therefore, a coalitional game can be effectively designed to manage the communications through the blockchain and predict miners' strategies.

Game Theory for Edge/Fog Computing

Incorporating mobile edge computing (MEC) technologies into blockchain-enabled IoT may potentially overcome resource constraints of smart end devices. Edge servers may store the whole blockchain and participate in most of heavy operations such as initiating and validating transactions while IoT devices may serve as lightweight nodes that only store partial blockchain data and undertake less computationally intensive tasks [263]. From a design perspective, game theory can assist in managing the interactions between MEC servers and client devices. For instance, Liu et al. [246] formulated the video transcoding and delivery problem as a three-stage Stackelberg game. The architecture consists of users, base stations, and video providers that all use their computational and communication resources to provide video streaming in a distributed and secure manner. Hence, by using the blockchain, the transcoding service could run without a central server.

Zuo et al. [264] formulated the interaction between the MEC server and users as a two-stage Stackelberg game in mobile blockchain networks. Hence, the achieved Stackelberg equilibrium enables the server to maximize its reward as well as the profit of mobile users. On the other hand, Xiong et al. [265] proposed a game-theoretic approach to analyze the interaction between miners and cloud/fog providers (CFP) in blockchain-based IoT. They presented a lightweight PoW-based blockchain so that the computation overhead is delegated to the cloud/fog. In addition, the management of computational resources in the blockchain consensus process is formulated as a Stackelberg game, through which the revenue earned by CFPs and the miners can be effectively optimized.

Guo et al. [266] proposed a mining task offloading approach using the Stackelberg game and double auction to exploit idle network resources and reduce communication overhead and delay. Ding et al. [267] proposed a model allowing the blockchain to encourage IoT nodes to participate in the mining process by purchasing more computational resources from edge servers. The interaction between IoT nodes and the blockchain is formulated as a

Stackelberg game, where the blockchain platform is the leader and the nodes are followers intending to maximize their profit. Xiong et al. [268] proposed a framework to maximize the gains of devices and the MEC server performing heavy tasks in blockchain-based IoT using the Stackelberg game. Miners must decide on a trade-off between the reward from mining and the price to purchase resources. Liu et al. [269] also proposed an architecture based on the Stackelberg game by formulating the interaction between blockchain users and miners. Here, IoT devices want to store their information in the blockchain as transactions.
Wenlong Guo et al. [270] proposed an optimal incentive scheme also based on the Stackelberg game to manage computational resources. This approach formulates the interaction between the edge service providers (ESP) as the leader and mining devices as followers. The ESP provides its computational resources to miners during mining. The game enables the miners and ESP to choose optimal strategies for allocating computational resources and maximizing their profits. This approach assumes three types of rewards: fixed reward, performance reward, and participant reward. The fixed reward is the constant reward for generating a new block. The performance reward is considered as a reward related to the size of the block and the volume of the transaction, while a participant reward depends on the level of participation of miners.
When miners try to mine new blocks, the reward is sometimes not adequate to encourage miners to keep mining. The blockchain with miners who are not incentivized to mine due to low rewards is called a gap chain, and the blockchain with miners who are encouraged to mine due to sufficient profits is called the normal chain. For the first time, Yuan et al. [271] proposed the normal gap game for managing edge resource allocation in the blockchain, where the gap and normal chain coexist. Here, miners compete with each other to maximize their profits. This approach allows the miners to follow the best strategy to manage their computational resources.
Ding et al. [272] proposed a framework for budget allocation in blockchain-based IoT systems with edge computing. The scenario considered consists of IoT resource-constrained devices that require edge servers to perform computational tasks including data analysis or storage. The interaction between buyers and sellers, IoT devices, and edge servers is formulated using a Stackelberg game model. To the best of our knowledge, there has been little investigation of using game-theoretic models to manage budget allocation in blockchain-based IoT systems, which creates a potential research direction.

Game Theory for Blockchain Operations

Anyone in the blockchain network can mine and verify the transactions to obtain profits. In order to gain the maximum rewards, each miner should analyze what strategy to adopt.

Game theory can be leveraged to fulfill this goal. Some of the applications of game theory to the blockchain operational phases are normally in computational resource management, achieving maximum reward, and nonce selection strategies.

Computational resource management

Each miner decides whether or not to allocate a share of its computational power based on the strategies taken by the other miners. Yang et al. [273] proposed to use non-cooperative game theory to achieve decentralized scheduling of multiple reusable energy sources in the energy local network (ELN) system. Each player in the game needs to make a decision to achieve the optimal objective under limited resources while considering players' interaction and the iterative strategy updates. This is done by formulating its own collaboration strategy to achieve the overall game goal. To deploy the decentralized automated demand response framework for energy storage in the ELN system, each decision maker should select an optimal consumption plan from its strategy set based on the available information.
Wang et al. [250] proposed a scheme based on blockchain in fog computing to optimize resource management using differential game theory. Fog nodes have limited computational resources; hence, they need to deploy the optimal management strategy and achieve maximum benefits. This game is an effective solution to represent the dynamic process of fog nodes' strategy generation. Yao et al. [274] formulate the problem of pricing and resource management between industrial IoT miners and cloud providers using a Stackelberg game, where the provider is the leader who sets the price first, then the miners act as followers and pick their strategies.

Maximizing reward

Bataineh et al. [275] presented a game-theoretic model exhibiting a mix of cooperative and competitive strategies between the miners and data providers to help them both determine the monetary reward and allocate computational resources. Chen et al. [276] designed a blockchain-based resource auction mechanism in a distributed edge storage scenario, where an auction model based on Bayesian Nash equilibrium is used to maximize profit. Furthermore, efficient pricing-based resource management for mobile mining in edge computing is presented by Xiong et al. [247] to encourage offloading operations in the blockchain platform. In particular, a Stackelberg game is proposed to maximize the utilities of miners as well as the edge computing provider.
Gao et al. [277] presented an approach to establish a Nash equilibrium for all parties in the supply chain using game-theoretic models. A bidding session is created, and the game is

played between a set of suppliers and retailers. Also, Zhang et al. [278] proposed an approach where the electricity consumer creates a contract including its trading strategies, whose goal is to attract small-scale electricity suppliers (SESs) to sell electricity and maximize their revenue. In addition, SESs get maximal rewards if they select the right contract of their own types. Finally, Jiang and Wu [98] used a game-theoretic model to study the effect of block size on miners' payoff. They defined a strategy for varying block sizes due to earning profits (e.g., determining the optimal default size). This approach can be effectively implemented in blockchain-based IoT systems.

Nonce selection strategy

Zuo et al. [264] formulated the user's nonce selection strategy as a non-cooperative game, where the utilities of the individual users are maximized in the untrusted MEC-aided mobile blockchain networks. This approach proves the existence of a Nash equilibrium and argues that cooperation behavior is unsuitable for blockchain-enabled IoT devices by using the repeated game concept.

Game Theory for Energy Consumption

Throughout the mining process, IoT devices with limited computational resources cannot satisfy the requirements of on-demand energy consumption. To tackle this issue, Li et al. [279] designed a decentralized, on-demand energy supply framework based on microgrids to satisfy the different energy demands of miners in response to consensus protocols. The Stackelberg game is used to formulate the energy allocation among microgrids and miners and achieve optimal profits for both. The microgrid is considered the game leader offering a nonuniform pricing strategy for miners, and a game is launched among the IoT devices that purchase energy from the microgrids. These perform real-time scheduling and decision making to enable the system to maximize the smoothness of operations, while every device intends to maximize its benefits.

Li et al. [280] designed a secure energy trading system in industrial IoT using the consortium blockchain. They proposed a credit-based payment approach to minimize transaction delay and support fast payment along with frequent energy trading. The approach also uses the Stackelberg game to achieve an optimal strategy for credit loans that maximizes the utility of credit banks. For energy trading in the internet of electric vehicles (IoEV), Ali et al. [281] proposed a framework based on the Stackelberg game and the IoTA blockchain to enable EVs to provide energy to vehicles and grids. The game allows the buyer to select the best seller to negotiate the energy price.

Game Theory for Consensus Algorithms

Many efficient consensus algorithms for blockchain networks have been presented. Many surveys exist on analyzing consensus algorithms in terms of scalability, how to reward validators, and security risks [282]. Game theory is also applied in the design of consensus mechanisms. For instance, Kumar and Jain [99] presented the consensus algorithm PoG that can be applied in IoT systems having limited-resource devices. PoG can manage the miners so that they can be restricted in creating the number of blocks based on their levels in the game and their rewards. This algorithm can be used for a single miner (i.e., one player) or a pool of miners (i.e., multi-player), and can be applied for resourceful and restricted-resource environments so that the number of game levels and their difficulty are based on resource availability in the network.

A game-theoretic model is also integrated into the fully decentralized consensus protocol to reward honest validators and punish dishonest or lazy ones that do not adhere to the protocol as a way of deterring attacks on the blockchain [283]. Also, a game-theoretic consensus protocol for resource-constrained devices (artificial medical body parts) was proposed in the field of healthcare, namely LPoG, to enable a secure and faster data-centric network [32]. Yaxing Wei et al. [284] proposed a new delegated proof of stake (DPoS) consensus algorithm and a game-theoretic reward and punishment incentive mechanism to improve the voting enthusiasm of blockchain entities and decrease the probability of occurrence of attacks.

Di et al. [285] used a Consensus Game (CG) in blockchain-based IoT to manage the data trading process. They formulated the interactions between IoT sensors as a game combining cooperative and non-cooperative models named “Consensus Game of IoT". The results demonstrate that such hybrid solutions can have an important role in blockchain over IoT. Finally, Alhasnawi et al. [286] presented an advanced demand management scheme based on coalitional game theory as a consensus algorithm to minimize the power mismatch, energy bill, and load energy waste. The communication packet loss can be reduced by using the consensus protocol

Game Theory for Data Sharing

IoT devices face the challenge of storage limitation and resource management as they create and collect large amounts of data. Jiang et al. [287] designed a blockchain-based data sharing framework for industrial IoT using game theory, where the relations between data owners and edge devices, such as service interactions and data storage, are formulated as a Stackelberg game. Zhang et al. [288] also proposed to solve the problem of data sharing and facilitate quality evaluation of the data created by IoT devices using a two-phase Stackelberg game

that attempts to maximize the profits of participants.
Bai et al. [289] enabled efficient cross-chain edge data sharing in blockchain-based Industrial Internet of Energy (IIoE). They used a two-step Stackelberg game to achieve the gains from resource scheduling while considering the preferences and risk factors of edge users. Si et al. [290] designed a lightweight blockchain-based information sharing system for IoT devices via a dynamic cooperative game to encourage nodes to behave honestly and mitigate malicious behavior.
Raifa Akkaoui et al. [291] designed a blockchain-based personal health data trading trust model and verification mechanism by leveraging an evolutionary game with two types of players: a data generator who is a patient wanting to share their data and be part of the Medical IoT (MIoT), and a data requester who is accountable for rewarding data generators to motivate them to share their data. There are four different strategy profiles: trustworthy patient, trustworthy requester, untrustworthy patient, and untrustworthy requester. Finally, Jiang and Wu [292] proposed to solve shortage limitations for mining devices in mobile environments using non-cooperative games. The interactions among miners are formulated to explore the impact of delegating mining mechanisms on miners' utilities. Overall, game theory has helped mobile miners to find optimal strategies to tackle storage limitation problems.

Discussion and Insights

This section investigates open challenges related to implementing the blockchain in IoT systems using game-theoretic designs and explores potential research directions in which game theory can play an important role. Some challenges mainly relate to IoT functional and architectural requirements in terms of performance and security. On the other hand, several open problems and limitations still exist when it comes to effectively and efficiently managing blockchain operations in IoT and practically implementing game-theoretic models.

Impact on IoT Performance and Security

Most of the existing consensus protocols such as PoW require intensive computations, where IoT nodes must execute computational tasks frequently, thereby leading to energy wastage. In addition, IoT nodes participating in the blockchain system should communicate and cooperate with other nodes, and each block added to the blockchain needs to be broadcast to all nodes, which results in communication overhead. However, most IoT devices have limitations in computation and communication capacity. Although game-theoretic models can be applied to allow IoT nodes to manage their computational and communication resources when the consensus algorithm is executed, finding an efficient game theory-based consensus

algorithm for IoT devices by saving on energy consumption and reducing computation overhead remains a challenge.

On the other hand, IoT devices using blockchain can become vulnerable to security threats, including selfish, majority, and DDoS attacks. Although these can be mitigated via game-theoretic approaches, various attacks such as eclipse [52] still need to be addressed, potentially using game-theoretical formulations. In IoT, the open nature of mining pools can attract a large number of miners for solving difficult problems together to improve efficiency. However, this increases the attack surface. Recent studies investigated the strategic trade-off that must be established between network openness and vulnerability in PoW-based blockchain [293]. Furthermore, ensuring data privacy when deploying the IoT over the blockchain is a critical challenge. Hence, more research is needed to investigate privacy-preserving ways when deciding on which transactions should be offloaded to the blockchain and how differential privacy can be combined with blockchain.

Open Challenges to Blockchain Operations

There exist several challenges related to blockchain design and they can be categorized as follows.

Security

Although the blockchain can mitigate security attacks against IoT, it introduces new security threats. For instance, malicious nodes participating in the blockchain may try to maximize their profits by deviating from the protocol. Also, blockchain systems may introduce new program vulnerabilities related to smart contracts. Game theory can be applied to mitigate the likelihood of attacks such as selfish mining by predicting anomalous behavior and adapting the defense strategies of the other miners in the pool. Such defense approaches should be extrapolated to address other attack types, such as Sybil and device spoofing. Although risk assessment has already been conducted for generic cyber applications of blockchain technology [294], a more tailored security risk assessment must be performed in the context of IoT, taking into consideration the confluence of its multiple dynamic characteristics, such as pervasiveness and cyber-physical operations. Stochastic games for risk modeling under uncertainty can serve as a breeding ground for developing more sophisticated approaches that emphasize the nonlinear, erratic, and unpredictable nature of vulnerability exploitation in networked dynamical systems driven by blockchain.

Blockchain architecture

The three types of blockchain, including public, private, and consortium, can be utilized in blockchain-based IoT systems. However, selecting an appropriate blockchain based on its features, advantages, and drawbacks is challenging. Tran et al. [239] surveyed several positions of blockchain in IoT systems. However, analyzing existing blockchain architectures in terms of performance and security should be considered. In addition, the number of blockchain networks used in the blockchain-based IoT architecture can have a significant impact on the performance of the whole system. Adopting a game-theoretic perspective to perform such an analysis can be a promising direction.

Consensus algorithms and energy consumption

Developing optimized consensus algorithms that fit the IoT architecture is another significant issue. For energy-constrained devices, improving consensus algorithms by using practical game-theoretic models can reduce energy consumption. To the best of our knowledge, there is no comprehensive work that compares and analyzes existing consensus algorithms with game theory-based designs in terms of energy consumption, computation, communication, and storage overhead.

Limitations of Game-theoretical Design

Incorporating game-theoretic models in blockchain-based IoT may also raise some challenges, as explained below.

Implementation

The deployment of game-theoretic models have not yet received its share of attention. In practice, implementing gamified approaches may entail some challenges. For instance, reaching an optimal decision by the IoT nodes may not be straightforward. Since the blockchain is decentralized, there is a large number of distributed nodes that participate in the blockchain network. Therefore, it is challenging to make optimal decisions under incomplete information where the leader nodes cannot observe the strategy of the follower nodes. Games that are based on partially-observable environments or the mean-field theory may be explored to cope with large-scale systems. In addition, due to the high computation overhead of game theory-based consensus algorithms, their practicality should be investigated in future research to ensure that deployment challenges can be addressed in practice. Finally, realism can be further improved by reducing the number of abstract game-theoretic parameters and evaluating

the game theory-enabled blockchain networks using real measured data and testbeds.

Nash equilibrium

Finding Nash equilibrium can be hard in a decentralized system like blockchain, which can have a large number of miners participating in the consensus protocol. IoT nodes have two choices when a Nash equilibrium exists: residing in the pool or leaving it. Similarly, the transaction fee may be paid by blockchain users when they choose to withdraw [248]. In the Nash equilibrium, players are not willing to deviate from the chosen strategies. However, more than one Nash equilibrium could be reached as the process of finding the best strategy is largely demanding for players [97].

Types of game models

This survey discussed several types of game-theoretic models used in blockchain-based IoT. However, selecting an adequate game model that fits the IoT architecture and the various design aspects of the blockchain constitutes a potential research direction. For instance, based on the target application area, designers can choose between simultaneous vs sequential play, as well as cooperative vs non-cooperative games. Nonetheless, we find that dynamic games are well-suited to solve many of the challenges discussed in this survey compared to one-shot games. Finally, most existing work has considered security games among mining pools, which might not be fully realistic in pervasive computing systems since the optimal strategy of the individual miner is not necessarily the optimal strategy for the overall pool. Designing a dynamic mean-field game to enable an individual node to make strategic decisions in an iterative fashion as part of the mining pool can be a potential solution. Mean-field game theory provides a powerful mathematical tool for problems with a large number of non-cooperative players. Since the subtle changes among nodes can be negligible if the number of players is sufficiently large, we usually research the epsilon Nash to analyze game stability [295].

Conclusion

This timely survey covered the recent research contributions made toward blockchain-based IoT systems using game-theoretical models. While the blockchain has significant advantages in establishing a secure platform for IoT applications, it entails some limitations that need to be emphasized during the design phase. The proposed taxonomy of game theory approaches will allow future researchers to address existing challenges in integrating the blockchain within

IoT systems and achieve smarter designs. Moreover, new design problems and solutions will likely emerge and contribute to this roadmap. In the future, we plan to explore other methodological avenues that will enable system designers to improve the security, efficiency, and management of blockchain platforms.

Acknowledgement

This research is partially supported by the Natural Sciences & Engineering Research Council of Canada (NSERC).